CRYSTALS – Dilithium: Digital Signatures from Module Lattices

Léo Ducas
CWI
Netherlands
ducas@cwi.nl

Tancrède Lepoint
SRI International
USA
tancrede.lepoint@sri.com

Vadim Lyubashevsky
IBM Research – Zurich
Switzerland
vad@zurich.ibm.com

Peter Schwabe
Radboud University
Netherlands
peter@cryptojedi.org

Gregor Seiler
IBM Research – Zurich
Switzerland
gse@zurich.ibm.com

Damien Stehlé
ENS de Lyon
France
damien.stehle@ens-lyon.fr

ABSTRACT

This paper presents Dilithium, a lattice-based signature scheme that is part of the CRYSTALS (Cryptographic Suite for Algebraic Lattices) package that will be submitted to the NIST call for post-quantum standards. The scheme is designed to be simple to securely implement against side-channel attacks and to have comparable efficiency to the currently best lattice-based signature schemes. Our implementation results show that Dilithium is competitive with lattice schemes of the same security level and outperforms digital signature schemes based on other post-quantum assumptions.

1 INTRODUCTION

Cryptography based on the hardness of lattice problems is seen as a very promising potential replacement of classical cryptography after the eventual emergence of quantum computing. In this paper, we describe a signature scheme that will be submitted as a candidate to the NIST call for new standards for digital signature, encryption, and key establishment protocols [19].

The goal for our design is to obtain a scheme that is simple to implement, efficient to run, and that is secure against lattice reduction attacks based on the same conservative parameter estimations as in [3]. The currently most efficient lattice-based scheme, BLISS [25], is based on the hardness of (a variant of) the NTRU problem and crucially uses discrete Gaussian sampling and high-precision rejection sampling to create compact signatures.

It has been recently shown (e.g., [17, 47]) that Gaussian sampling can create a lot of potential side-channel vulnerabilities that result in complete leakage of the secret key. While it is almost certainly possible to create good implementations which protect against (some) side-channel attacks (e.g., preventing timing attacks can be done following the approach in [43]), the intricacies involved make it an area in which one can very easily make mistakes. Since the goal is to create a scheme that is to be widely deployed, it would be highly desirable to have a scheme whose efficiency is comparable to that of BLISS, yet does not require Gaussian sampling.

It has also been shown that basing the hardness of cryptographic schemes on the NTRU problem (i.e., breaking the scheme is equivalent to finding a short vector in the NTRU lattice) may be weaker than expected, at least for large parameters [33]. While these attacks do not currently affect the parameters that are used in encryption and signature schemes, it does cause some concern that the problems may be somewhat easier than previously thought. Avoiding using the assumption that it is hard finding short vectors in the NTRU lattice is also therefore somewhat desirable if the efficiency penalty for doing so is not too high.

1.1 Our Proposal

Our scheme proposal, named Dilithium, is based on, and is an improvement of, the designs from [30] and [7]. The only sampling that is required in the scheme is uniform over some bounded domain and the rejection sampling part simply checks whether the individual coefficients of the potential signature are all smaller than a certain bound. The main improvement of Dilithium over [7, 30] is that the public key size is shrunk by more than a factor 2. We are able to do this by adapting the idea from [30] for producing "hints" about the carries caused by small summands (we in fact greatly simplify and improve the hint-generation from [30]). In [30], the idea for using "hints" was applied only to reduce the size of the signature, but we show that it can be used fairly effectively to also reduce the size of the public key.

While [7] was an improvement over [30] in that it removed the need for "hints" and therefore reduced the signature size by about 10%, reintroducing the hints allows us to significantly decrease the public key size at the expense of increasing the signature length by less than 5%. Since in many applications, the total sum of the public key size and the signature length is important, we believe that this is a very good trade-off. In fact, the total size of the public key plus signature size is only somewhat larger than that of BLISS for similar security levels. The parameter sizes for our signature scheme are presented in Table 1.

If instantiated so as to have similar security, the signature scheme BLISS would need to be implemented over a ring of dimension about double as what was used in [25], thus $\mathbb{Z}_q[X]/(X^{1024}+1)$. We would estimate that BLISS would have a public key and signature sizes of roughly 2KB and 1.5KB, respectively. The reason that the BLISS public key size would be larger than in our current construction, is because we do not see a way in which it can be compressed.

Our scheme was implemented on an Intel Core-i7 4770k (Haswell) processor and all the results are provided in Table 1. Further information and comparisons to schemes based on other post-quantum assumptions are detailed in Section 6.

We also present a variant of Dilithium that requires sampling from a discrete Gaussian distribution. We will refer to this variant as Dilithium-G. This scheme has slightly better parameters and slightly more security for the recommended parameter sizes. We designed Dilithium-G so that it works over the same ring as Dilithium. The

importance of this is that, while the signing algorithms of the two schemes are quite different, the verification algorithm is very similar and does not require any sampling. Thus it is possible to envision a scenario in which the signers could choose which signature to use because the verifiers would be able to verify signatures created by either signing algorithm. In particular, signers who are sure that their schemes would not be affected by side-channel attacks could use Dilithium-G, while those who create signatures in more hostile environments would use Dilithium.

1.2 Design Considerations

Our scheme, as well as all the ones mentioned above, are built via the "Fiat-Shamir with Aborts" framework [36] as illustrated in Fig. 1. The security of schemes built in this manner is based on the hardness of finding short vectors in lattices over the ring R_q. Past works have instantiated the scheme in one of two ways. Either they set the parameters $k = \ell = 1$ and $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ or they set $k, \ell > 1$ and $R_q = \mathbb{Z}_q$. The former choice results in a scheme based on the hardness of the Ring-LWE and Ring-SIS problems (or, in the case of BLISS, the NTRU problem). The latter choice of parameters makes the scheme based on the LWE / SIS problem. The general case where $k, \ell \geq 1$ and $R = \mathbb{Z}_q[X]/(X^n + 1)$ results in a scheme based on the Module-LWE / Module-SIS problem.

The main structural requirement that affects the parameter sizes is that the challenge c must have small norm and come from the ring R_q. Thus if R_q is a ring that does not have many elements of small norm (like $\mathbb{Z}_q$), one will either have large signatures due to the fact that c will be extremely large, or will choose a c from a smaller domain but need to repeat the scheme in parallel to decrease the soundness error, which will also serve to increase the signature size.[1]

The efficacy of lattice reduction attacks on the scheme is based on the products kn and ℓn, where n is the degree of the ring R_q. Thus based on the above, it is most efficient to set $k, \ell = 1$ and choose a large ring R_q. Our approach is to fix the ring $R_q = Z_q[X]/(X^{256}+1)$ and vary k, ℓ. For the size of the challenge space, this is a little sub-optimal. If, for example, we would take $k, \ell = 4$, then this is the same as using $k, \ell = 1$ and a ring of dimension 1024. In a ring of dimension 1024, one could have a subset of size 2^{256} all of whose elements have ℓ_2 norm less than 6. In a 256-dimensional ring, the ℓ_2 norm would be a little less than 8. Because the two numbers are rather close, this has a rather small influence on the parameters, and we find that the advantages of using a fixed ring outweighs this disadvantage.

A major advantage of choosing an $\mathbf{A} \in R_q^{k \times \ell}$ is the flexibility of the scheme. A large portion of the running time of the signing and the verification algorithms is performed doing multiplications in the ring $\mathbb{Z}_q[X]/(X^{256} + 1)$. Thus optimizing this part is crucial to getting the scheme to be efficient. Because the ring is exactly the same for every k, ℓ, one can easily re-use the critical parts of the implementation for schemes of various security levels. If one were to base the schemes on Ring-LWE / Ring-SIS, however, one would need to change rings in order to change security, which would require the re-implementation of all the routines. This could be especially costly in hardware. By keeping the ring constant and only varying the parameters k and ℓ, on the other hand, would make varying the security (whether due to novel cryptanalysis or for different application requirements) extremely simple. Being able to easily vary the security could also make it easier to adopt the scheme prior to the standardization process. Other advantages include the fact that the scheme has less algebraic structure, thus reducing the available practical attack avenues, as well as the fact that one can vary the parameters k and ℓ independently, which results in schemes that are less wasteful in setting the dimensions.

[1] There is also the option of defining the secret key to be a matrix rather than a vector, and the challenge c to be a vector [37]. This reduces the signature size, but increases the size of the secret and public keys.

Key Generation:

1: $\mathbf{A} \leftarrow R_q^{k \times \ell}$
2: $\mathbf{s}_1 \leftarrow S^{\ell}, \mathbf{s}_2 \leftarrow S^k$
3: $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \bmod q$.
4: Public Key $:= (\mathbf{A}, \mathbf{t})$
5: Secret Key $:= (\mathbf{s}_1, \mathbf{s}_2)$

Sign$(\mu, \rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t})$:

1: $\mathbf{y}_1 \leftarrow Y^{\ell}, \mathbf{y}_2 \leftarrow Y^k$
2: $\mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2 \bmod q$
3: $c := \mathrm{H}(\mathbf{A}, \mathbf{t}, \mathbf{w}, \mu)$
4: $\mathbf{z}_1 := \mathbf{y}_1 + c\mathbf{s}_1, \mathbf{z}_2 := \mathbf{y}_2 + c\mathbf{s}_2$
5: Run RejectionSample$(\mathbf{z}_1, \mathbf{z}_2, c\mathbf{s}_1, c\mathbf{s}_2)$, and goto 1 if it rejects
6: **output** $(\mathbf{z}_1, \mathbf{z}_2, c)$

Verify$(\mu, (\mathbf{z}_1, \mathbf{z}_2, c), \mathbf{A}, \mathbf{t})$:

1: $\mathbf{w} := \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} \bmod q$
2: **ACCEPT** if $c = \mathrm{H}(\mathbf{A}, \mathbf{t}, \mathbf{w}, \mu)$, and $\|(\mathbf{z}_1, \mathbf{z}_2)\|$ is small.

Figure 1: Fiat-Shamir with Aborts Framework. The sets S and Y are subsets of R with small coefficients. H is a cryptographic hash function outputting to a low-norm subset of R of size at least 2^{256}. The RejectionSample procedure in the signing algorithm makes the signature part $(\mathbf{z}_1, \mathbf{z}_2)$ independent of the secret key.

1.3 Security and Quantum Security

Our schemes are proven secure in the random oracle model, but we do not have a security proof in the quantum random oracle model. This is the same state of affairs as for every other somewhat *efficient* scheme using the Fiat-Shamir framework. It is possible to modify the parameters of our scheme to enable a security proof in the quantum random oracle model as done in [2] using the analysis from [1] for the scheme in [7], but this would increase certain parameter sizes by at least an order of magnitude. A different approach to get a lattice-based scheme secure in the quantum random oracle model would be to use the full-domain-hash design strategy as in [28]. A particularly efficient instantiation of the preceding scheme using NTRU lattices was given in [26]. The main problem is that to get small signatures would require sampling from a high-precision discrete Gaussian distribution with varying centers. This is an even more computationally intensive and intricate operation than the discrete Gaussian sampling required for BLISS. There are

work-arounds to not use the Gaussian distribution [41], but the result ends up significantly increasing the parameter sizes. One could also consider using lattice-based schemes that do not use any random oracles (e.g. [16, 27]), but those are even less efficient.

The lack of a security proof for our scheme in the quantum random oracle model is partially due to the same principle as why it is not possible to prove many classically-secure computationally-binding commitment schemes secure against a quantum adversary (see for example the discussion in [24]). In particular, we do not know how to discard the scenario in which the adversary produces a commitment together with some quantum state, which he can use to open the commitment to any value. Since the quantum state has been measured during the opening, the adversary cannot be asked to open the commitment to a second value and thus produce a collision, thus solving a presumably-hard problem upon which we wish to base the hardness of the commitment scheme. In short, a quantum adversary may theoretically be able to open the commitment only once, but do it to any value.

In Fiat-Shamir signatures, the first step of the protocol can be thought of as the commitment, the signature is the opening of a commitment, and the challenge is the committed message. Thus we cannot formally dismiss the scenario in which the adversary is able to come up with a commitment that he can open (i.e., sign) for a random message (the challenge). Nevertheless, as far as we know, there are no "natural" candidates of classically-secure commitment schemes that can be broken in this manner by a quantum adversary. Furthermore, Fiat-Shamir schemes that correspond to perfectly-binding commitment schemes have been shown to be secure in the quantum random oracle model [51], and the only examples of insecure Fiat-Shamir schemes are quite unnatural [4], which is somewhat similar to the status of the Fiat-Shamir heuristic in the classical world – it is known that insecure instances exist, but these instances are specifically tailored to be insecure [13, 29]. It is therefore also a reasonable assumption that "natural" Fiat-Shamir signatures that are secure in the random oracle model, and use "quantum-secure" cryptographic hash functions,[2] are also secure when the adversary has quantum access to the cryptographic hash function. Finding a proof for this conjecture, or refuting it with a counter-example, remains a major open problem.

2 PRELIMINARIES

For a set S, we write $s \leftarrow S$ to denote that s is chosen uniformly at random from S. If S is a probability distribution, then this denotes that s is chosen according to the distribution S.

All our algorithms are probabilistic. If b is a string, then $a \leftarrow \mathsf{A}(b)$ denotes the output of algorithm A when run on input b; if A is deterministic, then a is a fixed value and we write $a := \mathsf{A}(b)$.

[2]Proving security of Fiat-Shamir protocols based on the hardness of some problem P involves proving a statement of the form "If the Adversary succeeds, then either problem P is easy or finding a 2^{nd} pre-image in the cryptographic hash function H is easy." Thus we will be assuming that the H function used in the Fiat-Shamir signature is 2^{nd} pre-image-resistant against quantum algorithms. This requires doubling the output range of a "classically-secure" cryptographic hash function (e.g. SHA-3) to protect against Grover's algorithm.

2.1 Cryptographic Definitions

A signature scheme SIG = (KeyGen, Sign, Verify) is a triple of probabilistic polynomial-time algorithms together with a message space $\mathcal{M}$. The key-generation algorithm KeyGen returns a pair (pk, sk) consisting of a public key and a secret key. The signing algorithm Sign takes a secret key sk and a message $m \in \mathcal{M}$ to produce a signature Σ. Finally, the deterministic verification algorithm Verify takes a public key pk, a message $m \in \mathcal{M}$ and a signature Σ, and outputs either 0 (reject) or 1 (accept). We say that SIG is correct if for all message $m \in \mathcal{M}$ and all $(pk, sk) \leftarrow \mathsf{KeyGen}()$, we have $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

We recall the standard security notion for signature of strong existential unforgeability under chosen-message attacks (seu-CMA). The advantage of an adversary A is defined as $\mathbf{Adv}^{\mathrm{cma}}_{\mathrm{SIG}}(\mathsf{A}) =$

$$\Pr\left[b = 1 \;:\; \begin{array}{l}(sk, pk) \leftarrow \mathsf{KeyGen}();\\ (m^\star, \Sigma^\star) \leftarrow \mathsf{A}^{\textsc{Sign}(\cdot)}(pk);\\ b := \mathsf{Verify}(pk, m^\star, \Sigma^\star)\end{array}\right],$$

where the signing oracle is defined as $\textsc{Sign}(\cdot) := \mathsf{Sign}(sk, \cdot)$. We further require that $(m^\star, \Sigma^\star)$ is none of the input-output pairs A obtained through the $\textsc{Sign}(\cdot)$ queries. The scheme SIG is said to be (t, q_S, ϵ) seu-CMA *secure* if no adversary running in time at most t and making at most q_S queries to the signing oracle has advantage $\mathbf{Adv}^{\mathrm{cma}}_{\mathrm{SIG}}(\mathsf{A})$ greater than ϵ.

In the random oracle model [9], the adversary A is additionally given access to a random oracle $\mathsf{H}(\cdot)$ that it can query up to q_H times.

2.2 Rings and Distributions

We let R and R_q respectively denote the rings $\mathbb{Z}[X]/(X^n + 1)$ and $\mathbb{Z}_q[X]/(X^n + 1)$, for q an integer. Throughout this paper, the value of n will always be 256 and q will be the prime $8380417 = 2^{23} - 2^{13} + 1$. Regular font letters denote elements in R or R_q (which includes elements in $\mathbb{Z}$ and $\mathbb{Z}_q$) and bold lower-case letters represent column vectors with coefficients in R or R_q. By default, all vectors will be column vectors. Bold upper-case letters are matrices. For a vector $\mathbf{v}$, we denote by $\mathbf{v}^T$ its transpose.

Modular reductions. For an even (resp. odd) positive integer α, we define $r' = r \bmod^{\pm} \alpha$ to be the unique element r' in the range $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (resp. $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$) such that $r' = r \bmod \alpha$. We will sometimes refer to this as a *centered* reduction modulo q. For any positive integer α, we define $r' = r \bmod^{+} \alpha$ to be the unique element r' in the range $0 \leq r' < \alpha$ such that $r' = r \bmod \alpha$. When the exact representation is not important, we simply write $r \bmod \alpha$.

Sizes of elements. For an element $w \in \mathbb{Z}_q$, we write $\|w\|_\infty$ to mean $|w \bmod^{\pm} q|$. We now define the ℓ_∞ and ℓ_2 norms for $w = w_0 + w_1 X + \ldots + w_{n-1} X^{n-1} \in R$:

$$\|w\|_\infty = \max_i \|w_i\|_\infty, \quad \|w\| = \sqrt{\|w_0\|_\infty^2 + \ldots + \|w_{n-1}\|_\infty^2}.$$

Similarly, for $\mathbf{w} = (w_1, \ldots, w_k) \in R^k$, we define

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty, \quad \|\mathbf{w}\| = \sqrt{\|w_1\|^2 + \ldots + \|w_k\|^2}.$$

We will write S_η to denote all elements $w \in R$ such that $\|w\|_\infty \leq \eta$.

Extendable output function. Suppose that Sam is an extendable output function, that is a function on bit strings in which the output can be extended to any desired length. If we would like Sam to take as input x and then produce a value y that is distributed according to distribution S (or uniformly over a set S), we write $y \sim S := \mathsf{Sam}(x)$. It is important to note that this procedure is completely deterministic: a given x will always produce the same y. For simplicity we assume that the output distribution of Sam is perfect, whereas in practice Sam will be implemented using random oracles and produce an output that is statistically close to the perfect distribution.

Hashing. Let B_h denote the set of elements of R that have h coefficients that are either -1 or 1 and the rest are 0. We have $|B_h| = 2^h \cdot \binom{n}{h}$.

For our signature scheme, we will need a cryptographic hash function that hashes onto B_{60} (which has more than 2^{256} elements). The algorithm we will use to create a random element in B_{60} is sometimes referred to as an "inside-out" version of the Fisher-Yates shuffle.[3]

Algorithm 1 Create a random 256-element array with 60 ± 1's and 196 0's

Initialize $\mathbf{c} = c_0 c_1 \ldots c_{255} = 00 \ldots 0$
for $i := 196$ to 255 **do**
  $j \leftarrow \{0, 1, \ldots, i\}$
  $s \leftarrow \{0, 1\}$
  $c_i := c_j$
  $c_j := (-1)^s$
end for
return $\mathbf{c}$

Therefore to create a function $\mathrm{H} : \{0,1\}^* \rightarrow B_{60}$, one would use the XOF Sam to expand the input to produce the randomness needed by Algorithm 1 – that is produce 60 j's, where j_i is uniformly random between 0 and i, and 60 bits for the s.

2.3 Module-LWE and Module-SIS

Let ℓ be a positive integer parameter. The hard problems underlying the security of our schemes are Module-LWE and Module-SIS, which were studied in [35], and are a generalization of the Ring-LWE [40] and Ring-SIS problems [38, 46].

Module-LWE distribution. The Module-LWE distribution is the distribution on $R_q^k \times R_q$ induced by pairs $(\mathbf{a}_i, b_i)$ where $\mathbf{a}_i \leftarrow R_q^\ell$ is uniform and $b = \mathbf{a}_i^T \mathbf{s} + e_i$ with $\mathbf{s} \leftarrow S_\eta^\ell$ common to all samples and $e_i \leftarrow S_\eta$ fresh for every sample.

Module-LWE. Module-LWE consists in recovering $\mathbf{s}$ from polynomially many samples chosen from the Module-LWE distribution. More precisely, for an algorithm A, we define $\mathbf{Adv}^{\mathrm{mlwe}}_{k,\ell,\eta}(\mathrm{A}) =$

$$\Pr\left[\mathbf{x} = \mathbf{s} : \begin{array}{l} \mathbf{A} \leftarrow R_q^{k\times\ell}; (\mathbf{s}, \mathbf{e}) \leftarrow S_\eta^\ell \times S_\eta^k; \\ \mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}; \mathbf{x} \leftarrow \mathrm{A}(\mathbf{A}, \mathbf{b}); \end{array}\right].$$

[3] Normally, the algorithm should begin at $i = 0$, but since there are 196 0's, the first 195 iterations would just be setting components of $\mathbf{c}$ to 0.

We say that the (t, ϵ) Module-$\mathrm{LWE}_{k,\ell,\eta}$ hardness assumption holds if no algorithm A running in time at most t has an advantage greater than ϵ.

(Inhomogeneous) Module-SIS. The Inhomogeneous Module-SIS problem consists in finding a pre-image $\mathbf{x}$ satisfying $[\ \mathbf{A} \mid \mathbf{I}\ ] \cdot \mathbf{x} = \mathbf{t}$, where $\mathbf{t} \leftarrow R_q^k$, $\mathbf{A} \leftarrow R_q^{k\times\ell}$ and $\mathbf{I}$ is the $k \times k$ identity matrix.[4] More precisely, for an algorithm A, we define $\mathbf{Adv}^{\mathrm{msis}}_{k,k+\ell,\beta}(\mathrm{A}) =$

$$\Pr\left[b = 1 \ : \begin{array}{l} \mathbf{A} \leftarrow R_q^{k\times\ell}; \\ \mathbf{t} \leftarrow R_q^k; \\ \mathbf{x} \leftarrow \mathrm{A}(\mathbf{A}); \\ b := (\mathbf{x} \in R^{k+\ell}) \wedge ([\ \mathbf{A} \mid \mathbf{I}\ ] \cdot \mathbf{x} = \mathbf{t}) \wedge (\|\mathbf{x}\| \le \beta) \end{array}\right].$$

We say that the (t, ϵ) Module-$\mathrm{SIS}_{k,k+\ell,\beta}$ hardness assumption holds if no algorithm A running in time at most t has an advantage greater than ϵ. We define Module-$\mathrm{SIS}^\infty_{k,k+\ell,\beta}$ as the direct adaptation to the infinity norm. The homogeneous version of the Module-SIS problem is defined with the target $\mathbf{t} = \mathbf{0}$ and the solution $\mathbf{x} = \mathbf{0}$ being disallowed.

3 ROUNDING ALGORITHMS

In this section, we introduce the rounding algorithms that we will use to compress the public key and signature in Dilithium.

3.1 High-order and Low-order Bits

We will break up elements in $\mathbb{Z}_q$ into their "high-order" bits and "low-order" bits in two ways. The first algorithm $\mathsf{Power2Round}_q$ is the straightforward bit-wise way to break up an element $r = r_1 \cdot 2^d + r_0$ where $r_0 = r \bmod^\pm 2^d$ and $r_1 = (r - r_0)/2^d$, and is presented in Algorithm 2.

Algorithm 2 $\mathsf{Power2Round}_q(r, d)$

$r := r \bmod^+ q$
$r_0 := r \bmod^\pm 2^d$
return $(r - r_0)/2^d$

Notice that if we choose the representatives of r_1 to be non-negative integers between 0 and $\lfloor q/2^d \rfloor$, then the distance (modulo q) between any two $r_1 \cdot 2^d$ and $r_1' \cdot 2^d$ is usually $\ge 2^d$, except for the border case. In particular, the distance modulo q between $\lfloor q/2^d \rfloor \cdot 2^d$ and 0 could be very small. For our applications, we would like to have a rounding procedure that keeps this distance bounded from below by $\approx 2^d$; this will allow us to have 1-bit hints for the uniform digital signature scheme and slightly decrease the hint size for the Gaussian one.

We accomplish this by selecting an α that is a divisor of $q-1$ and write $r = r_1 \cdot \alpha + r_0$ in the same way as before. For the sake of simplicity, we assume that α is even (which is possible, as q is odd). The possible $r_1 \cdot \alpha$'s are now $\{0, \alpha, 2\alpha, \ldots, q-1\}$. Note that the distance between $q-1$ and 0 is 1, and so we remove $q-1$ from the set of possible $r_1 \cdot \alpha$'s, and simply round the corresponding r's to 0. Because $q-1$ and 0 differ by 1, all this does is possibly increase the magnitude

[4] This is often referred to as the "Hermite Normal Form" of the problem. It is equivalent to the Module-SIS problem where the matrix $[\ \mathbf{A} \mid \mathbf{I}\ ]$ is replaced by a completely random matrix $\mathbf{A}' \leftarrow R_q^{k\times(k+\ell)}$.

of the remainder r_0 by 1. This procedure is called $\mathsf{Decompose}_q$ and is presented in Algorithm 3. For notational convenience, we also define $\mathsf{HighBits}_q$ and $\mathsf{LowBits}_q$ routines that simply extract r_1 and r_0, respectively, from the output of $\mathsf{Decompose}_q$; cf. Algorithms 4 and 5.

Algorithm 3 $\mathsf{Decompose}_q(r, \alpha)$

$r \coloneqq r \bmod^+ q$
$r_0 \coloneqq r \bmod^\pm \alpha$
if $r - r_0 = q - 1$ **then**
  $r_1 \coloneqq 0$
  $r_0 \coloneqq r_0 - 1$
else
  $r_1 \coloneqq (r - r_0)/\alpha$
end if
return (r_1, r_0)

Algorithm 4 $\mathsf{HighBits}_q(r, \alpha)$

$(r_1, r_0) \coloneqq \mathsf{Decompose}_q(r, \alpha)$
return r_1

Algorithm 5 $\mathsf{LowBits}_q(r, \alpha)$

$(r_1, r_0) \coloneqq \mathsf{Decompose}_q(r, \alpha)$
return r_0

Finally, when these algorithms are called with $r \in R_q$ or $\mathbf{r} \in R_q^k$, the rounding procedure is applied to each coefficient individually.

3.2 Producing "Hints" for Dilithium

Let q and α be positive integers with $q > 2\alpha$ and $q \equiv 1 \pmod{\alpha}$ (and α even). Given $r, z \in \mathbb{Z}_q$ with $\|z\|_\infty \le \alpha/2$, we would like to produce a 1-bit hint y, such that one can derive $\mathsf{HighBits}_q(r + z, \alpha)$ from r, q, α and y. We propose a procedure $\mathsf{MakeHint}_q$ in Algorithm 6 to produce such a hint, and a procedure $\mathsf{UseHint}_q$ in Algorithm 7 that shows how to use the hint y to recover the higher order bits of $r + z$. When these algorithms are called with $r, z \in R_q$ or $\mathbf{r}, \mathbf{z} \in R_q^k$, the rounding procedure is applied to each coefficient individually.

Algorithm 6 $\mathsf{MakeHint}_q(z, r, \alpha)$

$r_1 \coloneqq \mathsf{HighBits}_q(r, \alpha)$
$v_1 \coloneqq \mathsf{HighBits}_q(r + z, \alpha)$
if $r_1 = v_1$ **then**
  return 0
else
  return 1
end if

Lemma 3.1. *Let $r, z \in \mathbb{Z}_q$ with $\|z\|_\infty \le \alpha/2$. Then*

$$\mathsf{UseHint}_q\left(\mathsf{MakeHint}_q(z, r, \alpha), r, \alpha\right) = \mathsf{HighBits}_q(r + z, \alpha).$$

Algorithm 7 $\mathsf{UseHint}_q(y, r, \alpha)$

$m \coloneqq (q - 1)/\alpha$
$(r_1, r_0) \coloneqq \mathsf{Decompose}_q(r, \alpha)$
if $y = 1$ and $r_0 > 0$ **then**
  return $(r_1 + 1) \bmod^+ m$
else if $y = 1$ and $r_0 \le 0$ **then**
  return $(r_1 - 1) \bmod^+ m$
else
  return r_1
end if

Proof. The output of $\mathsf{Decompose}_q$ is an integer r_1 such that $0 \le r_1 < (q - 1)/\alpha$ and another integer r_0 such that $\|r_0\|_\infty \le \alpha/2$. Because $\|z\|_\infty \le \alpha/2$, the integer $v_1 \coloneqq \mathsf{HighBits}_q(r + z, \alpha)$ either stays the same as r_1 or becomes $r_1 \pm 1$ modulo $m = (q-1)/\alpha$. More precisely, if $r_0 > 0$, then $-\alpha/2 < r_0 + z \le \alpha$. This implies that v_1 is either r_1 or $r_1 + 1 \bmod m$. If $r_0 \le 0$, then we have $-\alpha \le r_0 + z \le \alpha/2$. In this case, we have $v_1 = r_1$ or $r_1 - 1 \bmod m$.

The $\mathsf{MakeHint}_q$ routine checks whether $r_1 = v_1$ and outputs 0 if this is so, and 1 if $r_1 \ne v_1$. The $\mathsf{UseHint}_q$ routine uses the "hint" y to either output r_1 (if $y = 0$) or, depending on whether $r_0 > 0$ or not, output either $r_1 + 1 \bmod^+ m$ or $r_1 - 1 \bmod^+ m$. □

The lemma below shows that r is not too far away from the output of the $\mathsf{UseHint}_q$ algorithm. This will be necessary for the security of the scheme.

Lemma 3.2. *Let $(y, r) \in \{0, 1\} \times \mathbb{Z}_q$ and let $v_1 = \mathsf{UseHint}_q(y, r, \alpha)$. If $y = 0$, then $\|r - v_1 \cdot \alpha\|_\infty \le \alpha/2$; else $\|r - v_1 \cdot \alpha\|_\infty \le \alpha + 1$.*

Proof. Let $(r_1, r_0) \coloneqq \mathsf{Decompose}_q(r, \alpha)$. We go through all three cases of the $\mathsf{UseHint}_q$ procedure.

Case 1 ($y = 0$): We have $v_1 = r_1$ and

$$r - v_1 \cdot \alpha = r_1 \cdot \alpha + r_0 - r_1 \cdot \alpha = r_0\,,$$

which by definition has absolute value at most $\alpha/2$.

Case 2 ($y = 1$ and $r_0 > 0$): We have $v_1 = r_1 + 1 - \kappa \cdot (q - 1)/\alpha$ for $\kappa = 0$ or 1. Thus

$$\begin{aligned} r - v_1 \cdot \alpha &= r_1 \cdot \alpha + r_0 - (r_1 + 1 - \kappa \cdot (q-1)/\alpha) \cdot \alpha \\ &= -\alpha + r_0 + \kappa \cdot (q - 1). \end{aligned}$$

After centered reduction modulo q, the latter has magnitude $\le \alpha$.

Case 3 ($y = 1$ and $r_0 \le 0$): We have $v_1 = r_1 - 1 + \kappa \cdot (q - 1)/\alpha$ for $\kappa = 0$ or 1. Thus

$$\begin{aligned} r - v_1 \cdot \alpha &= r_1 \cdot \alpha + r_0 - (r_1 - 1 + \kappa \cdot (q-1)/\alpha) \cdot \alpha \\ &= \alpha + r_0 - \kappa \cdot (q - 1). \end{aligned}$$

After centered reduction modulo q, the latter quantity has magnitude $\le \alpha + 1$. □

The next lemma will play a role in proving the strong existential unforgeability of our signature scheme. It states that two different y, y' cannot lead to $\mathsf{UseHint}_q(y, r, \alpha) = \mathsf{UseHint}_q(y', r, \alpha)$.

Lemma 3.3. *Let $r \in \mathbb{Z}_q$ and $y, y' \in \{0, 1\}$. If $\mathsf{UseHint}_q(y, r, \alpha) = \mathsf{UseHint}_q(y', r, \alpha)$, then $y = y'$.*

Proof. Note that $\mathsf{UseHint}_q(0, r, \alpha) = r_1$ and $\mathsf{UseHint}_q(1, r, \alpha)$ is equal to $(r_1 \pm 1) \bmod^+ (q-1)/\alpha$. Since $(q-1)/\alpha \geq 2$, we have that $r_1 \neq (r_1 \pm 1) \bmod^+ (q-1)/\alpha$. □

The following lemma is a direct consequence of Lemmas 3.1 to 3.3.

Lemma 3.4. *Suppose that q and α are positive integers satisfying $q > 2\alpha$, $q \equiv 1 \pmod{\alpha}$ and α even. Let $\mathbf{r}$ and $\mathbf{z}$ be vectors of elements in R_q where $\|\mathbf{z}\|_\infty \leq \alpha/2$, and let $\mathbf{y}, \mathbf{y}'$ be vectors of bits. Then the $\mathsf{HighBits}_q$, $\mathsf{MakeHint}_q$, and $\mathsf{UseHint}_q$ algorithms satisfy the following properties:*

(1) $\mathsf{UseHint}_q(\mathsf{MakeHint}_q(\mathbf{z}, \mathbf{r}, \alpha), \mathbf{r}, \alpha) = \mathsf{HighBits}_q(\mathbf{r} + \mathbf{z}, \alpha)$.
(2) *Let $\mathbf{v}_1 = \mathsf{UseHint}_q(\mathbf{y}, \mathbf{r}, \alpha)$. Then $\|\mathbf{r} - \mathbf{v}_1 \cdot \alpha\|_\infty \leq \alpha + 1$. Furthermore, if the number of 1's in $\mathbf{y}$ is ω, then all except at most ω coefficients of $\mathbf{r} - \mathbf{v}_1 \cdot \alpha$ will have magnitude at most $\alpha/2$ after centered reduction modulo q.*
(3) *For any $\mathbf{y}, \mathbf{y}'$, if $\mathsf{UseHint}_q(\mathbf{y}, \mathbf{r}, \alpha) = \mathsf{UseHint}_q(\mathbf{y}', \mathbf{r}, \alpha)$, then $\mathbf{y} = \mathbf{y}'$.*

4 DILITHIUM

The scheme is built via the "Fiat-Shamir with Aborts" idea [36] and resembles the Bai-Galbraith scheme [7]. The main difference is that we compress the public key by a factor of a little larger than 2 and this requires using the algorithms from Section 3 to create and use hints for reconstructing high order bits of polynomials.

The parameters. The secret keys are chosen to have coefficients of magnitude at most η. We set β integer so that we have $\|s \cdot c\|_\infty < \beta$ with high probability (around $1 - 2^{-80}$) over the choices of $s \leftarrow S_\eta$ and $c \leftarrow B_{60}$. The parameters γ_1 and γ_2 dictate the rejection probability (see Equation (3)) and the size of the signatures: the larger the γ_i's, the larger the signature, but the smaller the probability of rejection.

4.1 The Signature Scheme

Our signature algorithm is described in Algorithms 8 to 10.

Algorithm 8 KeyGen()

1: $\rho, \rho' \leftarrow \{0,1\}^{256}$
2: $\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{Sam}(\rho)$
3: $(\mathbf{s}_1, \mathbf{s}_2) \sim S_\eta^\ell \times S_\eta^k := \mathsf{Sam}(\rho')$
4: $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
5: $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
6: **return** $(pk = (\rho, \mathbf{t}_1), sk = (\rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}))$

The key generation proceeds by choosing a random 256-bit seed ρ and expanding into a matrix $\mathbf{A} \in R_q^{k \times \ell}$ by an extendable output function Sam modeled as a random oracle. The secret keys $\mathbf{s}_1, \mathbf{s}_2$ are generated from Sam expanding a random seed ρ' and have uniformly random coefficients between $-\eta$ and η (inclusively). The value $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ is computed. The secret key is $\rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}$, while the public key is $\rho, \mathbf{t}_1$ with $\mathbf{t}_1$ output by Algorithm 2 (we have $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$ for some small $\mathbf{t}_0$).

The signing procedure starts by splitting the entire $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ into $\mathbf{t}_1$ and $\mathbf{t}_0$ such that $\mathbf{t}_1 \cdot 2^d + \mathbf{t}_0 = \mathbf{t}$, where $\|\mathbf{t}_0\|_\infty \leq 2^{d-1}$.

Algorithm 9 Sign($sk = (\rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}), \mu \in \mathcal{M}$)

1: $\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{Sam}(\rho)$
2: $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
3: $\mathbf{t}_0 := \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
4: $r \leftarrow \{0,1\}^{256}$
5: $\mathbf{y} \sim S_{\gamma_1 - 1}^\ell := \mathsf{Sam}(r)$
6: $\mathbf{w} := \mathbf{A}\mathbf{y}$
7: $\mathbf{w}_1 := \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$
8: $c := \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$
9: $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
10: $(\mathbf{r}_1, \mathbf{r}_0) := \mathsf{Decompose}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
11: **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$ **then** goto 4
12: $\mathbf{h} := \mathsf{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
13: **if** $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$ or the number of 1's in $\mathbf{h}$ is greater than ω **then** goto 4
14: **return** $\sigma := (\mathbf{z}, \mathbf{h}, c)$

The next step of the signing algorithm has the signer sample $\mathbf{y}$ with coefficients in $S_{\gamma_1 - 1}$ (using the extendable output function Sam and a random seed r) and then compute $\mathbf{w} = \mathbf{A}\mathbf{y}$. Then the signer writes $\mathbf{w} = 2\gamma_2 \cdot \mathbf{w}_1 + \mathbf{w}_0$, with $\mathbf{w}_0$ between $-\gamma_2$ and γ_2 (inclusively), and computes $c = \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu) \in B_{60}$ using the function introduced in Section 2.2.[5] After obtaining c, the signer computes $\mathbf{z} = \mathbf{y} + c\mathbf{s}$. If some coefficient of $\mathbf{z}$ is at least $\gamma_1 - \beta$, then the signing procedure restarts. The process also restarts if the magnitude of some coefficient of $\mathbf{r}_0 = \mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$ is at least $\gamma_2 - \beta$. This part of the protocol is necessary for security—it makes sure that nothing about the secret key $\mathbf{s}_1, \mathbf{s}_2$ is leaked. The last check makes sure that $\mathbf{r}_1 = \mathbf{w}_1$ and this is necessary for correctness. We should point out that if $\|c\mathbf{s}_2\|_\infty \leq \beta$, then $\|\mathbf{r}_0\|_\infty$ being less than $\gamma_2 - \beta$ immediately implies that $\mathbf{r}_1 = \mathbf{w}_1$. Since we want $\|c\mathbf{s}_2\|_\infty \leq \beta$ to be true with overwhelming probability for security, the probability that this last check will be useful is negligible. Still, we include it just to make the probability of a verifier accepting a valid signature being 1, rather than negligibly close to 1.

If all the checks pass and a restart is not necessary, then it can be shown (see Section 4.2) that $\mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2) = \mathbf{w}_1$. At this point, if the verifier knew the entire element $\mathbf{t}$ and $(\mathbf{z}, c)$, he could have recovered $\mathbf{w}_1$ and checked that $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $c = \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$. However, since we want to compress the public key, the verifier only knows $\mathbf{t}_1$. Hence, the signer needs to provide a "hint" which will allow the verifier to compute $\mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$. This is done in Step 12. In Step 13, the verifier performs some checks that will fail very rarely (much less than 1% of the time) that do not really affect the total running time. Most importantly, Step 12 is for compression and has no effect on the security of the scheme under the assumption that the verifier has the entire key $\mathbf{t}$ (thus the actual scheme may even be more secure in practice).

[5] Note that all these operations feature some trade-off between storage and computation time. For example, to optimize the computation time, one can store $\mathbf{t}_0$ in the secret key and can "partially compute" $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ by storing the part that depends on ρ and $\mathbf{t}_1$ as part of the signing key (and this way storing $\mathbf{t}_1$ is no longer necessary). On the other hand, if one is concerned about the secret key size, one can define $sk = (\rho, \rho')$ and recompute everything when signing.

Algorithm 10 Verify($pk = (\rho, \mathbf{t}_1), \mu \in \mathcal{M}, \sigma = (\mathbf{z}, \mathbf{h}, c)$)

1: $\mathbf{A} \coloneqq \mathsf{Sam}(\rho)$
2: $\mathbf{w}_1 \coloneqq \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
3: **if** $c = H(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ and $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and the number of 1's in $\mathbf{h}$ is $\leq \omega$ **then**
4: **return** 1
5: **else**
6: **return** 0
7: **end if**

Finally, verification works by using the signature and the public key to reconstruct the $\mathbf{w}_1$ and then check that $c = \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ and that all the coefficients of $\mathbf{z}_1$ are less than $\gamma_1 - \beta$, and that the number of 1's in $\mathbf{h}$ is no greater than ω. The number of ones in $\mathbf{h}$ is determined by how many values of $c\mathbf{t}_0$ cause a carry to occur. Since $c\mathbf{t}_0$ is not too large, there is a significantly larger probability that a carry does not occur. Experimentally with the parameters we recommend in Table 1, we computed the upper-bounds such that the number of carries is not larger than ω with very high probability (much larger than 99%).

Concrete parameters. We provide concrete parameters and security estimates in Table 1.

The public key consists of ρ and $\mathbf{t}_1$, which are 32 and $32 \cdot k \cdot (\lceil \log(q) \rceil - d)$ bytes, respectively. The signature consists of $\mathbf{z}$, which is $32 \cdot \ell \cdot \lceil \log(2\gamma_1) \rceil$ bytes, the polynomial $\mathbf{c}$ which can be represented with 40 bytes (32 bytes to give the position of the non-zero coefficients and 8 bytes to distinguish between the 1's and the -1's). The vector $\mathbf{h}$ contains $256 \cdot k$ binary coefficients, ω of which are 1's. The information-theoretic lower bound for the number of bits to represent such a vector is $\log \binom{256 \cdot k}{\omega}$ bits, but such a representation would be costly to obtain.

Instead, we store $\mathbf{h}$ as follows. We break up $\mathbf{h}$ into k vectors $\mathbf{h}_1, \ldots, \mathbf{h}_k$ of dimension 256. The position of the 1's in each $\mathbf{h}_i$ can be represented with 1 byte each. Since there are ω 1's, this amounts to ω bytes. Furthermore, we need to specify the separation between the k vectors. Since there are k vectors, there are $k - 1$ separator. One needs $\lceil \log \omega \rceil$ bits to represent each separation. Since $\omega < 128$ in all our instantiations, this amounts to 7 bits, which we round up to 1 byte for convenience. Thus the total cost (in bytes) to represent $\mathbf{h}$ is $\omega + k - 1$. One could use slightly more compact representations and possibly save around 20 bytes in the representation of $\mathbf{h}$. Nevertheless, we use the method above for the convenience of having each position be represented by exactly 1 byte.

4.2 Correctness

In this section, we prove the correctness of the signature scheme. We use the notation of Algorithms 8 to 10.

If $\|c\mathbf{t}_0\|_\infty < \gamma_2$, then by Lemma 3.4 we know that

$$\mathsf{UseHint}_q(\mathbf{h}, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\,.$$

Since $\mathbf{w} = \mathbf{Ay}$ and $\mathbf{t} = \mathbf{As}_1 + \mathbf{s}_2$, we have that

$$\mathbf{w} - c\mathbf{s}_2 = \mathbf{Ay} - c\mathbf{s}_2 = \mathbf{A}(\mathbf{z} - c\mathbf{s}_1) - c\mathbf{s}_2 = \mathbf{Az} - c\mathbf{t}, \tag{1}$$

and $\mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0 = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d$. Therefore the verifier computes

$$\begin{aligned}&\mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)\\ &= \mathsf{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\,.\end{aligned}$$

Furthermore, because the Signer also checks in Line 11 that $\mathbf{r}_1 = \mathbf{w}_1$, this is equivalent to

$$\mathsf{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2). \tag{2}$$

Therefore, the $\mathbf{w}_1$ computed by the verifier is the same as that of the signer, and the verification procedure will always accept.

4.3 Zero-Knowledge and Simulation

For the sake of simplicity, in this proof (as well as in the reduction in Section 4.5), we will be assuming that the public key is $\mathbf{t}$ rather than $\mathbf{t}_1$. The fact that $\mathbf{t}_0$ is not part of the genuine public key is not used anywhere in the proof.

We want to first compute the probability that some particular $(\mathbf{z}, c)$ is generated in Step 9 taken over the randomness of $\mathbf{y}$ and the random oracle H which is modeled as a random function. We have

$$\Pr[\mathbf{z}, c] = \Pr[c] \cdot \Pr[\mathbf{y} = \mathbf{z} - c\mathbf{s}_1 \mid c].$$

Whenever $\mathbf{z}$ has all its coefficients less than $\gamma_1 - \beta$ then the above probability is exactly the same for every such tuple $(\mathbf{z}, c)$. This is because $\|c\mathbf{s}_i\|_\infty \leq \beta$ (with overwhelming probability), and thus $\|\mathbf{z} - c\mathbf{s}_1\|_\infty \leq \gamma_1 - 1$, which is a valid value of $\mathbf{y}$. Therefore, if we only output $\mathbf{z}$ when all its coefficients have magnitudes less than $\gamma_1 - \beta$, then the resulting distribution will be uniformly random over $S^\ell_{\gamma_1 - \beta - 1} \times B_{60}$.

The simulation of the signature follows [7, 37]. The simulator picks a uniformly random $(\mathbf{z}, c)$ in $S^\ell_{\gamma_1 - \beta - 1} \times B_{60}$, after which it also makes sure that

$$\|\mathbf{r}_0\|_\infty = \|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta.$$

By Equation (1), we know that $\mathbf{w} - c\mathbf{s}_2 = \mathbf{Az} - c\mathbf{t}$, and therefore the simulator can perfectly simulate this as well.

If $\mathbf{z}$ does indeed satisfy $\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta$, then as long as $\|c\mathbf{s}_2\|_\infty \leq \beta$, we will have

$$\mathbf{r}_1 = \mathsf{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2) = \mathbf{w}_1.$$

Since our β was chosen such that the probability (over the choice of $c, \mathbf{s}_2$) that $\|c\mathbf{s}_2\|_\infty < \beta$ is very close to 1, the simulator does not need to perform the check that $\mathbf{r}_1 = \mathbf{w}_1$ and can always assume that it passes (we discuss the effect that this has on actual security in Section 4.4).

We can then program

$$\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu) \leftarrow c\,.$$

Unless we have already set the value of $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ to something else, the resulting pair $(\mathbf{z}, c)$ has the same distribution as in a genuine signature of μ. If $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ was previously assigned a value, then it must be that

$$\mathsf{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2) = \mathbf{w}_1 = \mathbf{w}'_1 = \mathsf{HighBits}_q(\mathbf{Az}' - c'\mathbf{t}, 2\gamma_2)$$

for some previously-chosen $\mathbf{z}', c'$. The above implies that there exist vectors $\mathbf{e}, \mathbf{e}'$ with $\|\mathbf{e}\|_\infty, \|\mathbf{e}'\|_\infty \leq \gamma_2 + 1$, such that $\mathbf{A}(\mathbf{z} - \mathbf{z}') + (\mathbf{e} - \mathbf{e}') = (c - c')\mathbf{t}$. Thus if $\mathbf{z} \neq \mathbf{z}'$ or $c \neq c'$ (which happens with probability greater than $1 - 2^{-10000}$ even for our smallest parameter set), we

Table 1: Parameters for Dilithium.

	weak	medium	recommended	very high
q	8380417	8380417	8380417	8380417
d	14	14	14	14
weight of c	60	60	60	60
$\gamma_1 = (q-1)/16$	523776	523776	523776	523776
$\gamma_2 = \gamma_1/2$	261888	261888	261888	261888
(k, ℓ)	(3, 2)	(4, 3)	(5, 4)	(6, 5)
η	7	6	5	3
β	330	285	235	145
ω	64	80	96	120
pk size $= 32 \cdot k \cdot (\lceil \log(q) \rceil - d) + 32$ bytes	896	1184	1472	1760
sig size $= 32 \cdot \ell \cdot \lceil \log(2\gamma_1) \rceil + (\omega + k - 1) + 40$ bytes	1386	2043	2700	3365
Repetitions (from Eq. (4))	3.65	4.65	5	3.35
BKZ block-size b to break SIS	235	355	475	605
Best Known Classical bit-cost	68	103	138	176
Best Known Quantum bit-cost	62	94	125	160
Best Plausible bit-cost	48	73	98	125
BKZ block-size b to break LWE	200	340	485	595
Best Known Classical bit-cost	58	100	141	174
Best Known Quantum bit-cost	53	91	128	158
Best Plausible bit-cost	41	71	100	124
KeyGen cycles	173, 100	325, 746	522, 992	723, 974
Sign cycles	741, 844	1, 530, 912	2, 253, 378	1, 856, 048
Verify cycles	239, 572	413, 644	625, 968	871, 190
KeyGen cycles (AVX2 optimized)	97, 544	162, 670	251, 590	323, 112
Sign cycles (AVX2 optimized)	404, 298	637, 994	1, 042, 250	838, 428
Verify cycles (AVX2 optimized)	131, 316	205, 684	297, 590	402, 738

have found the same type of solution (actually, an even slightly harder-to-find one) as the one in Lemma 4.1 upon which the security of our scheme rests.[6]

All the other steps (after Step 11) of the signing algorithm are performed using public information and are therefore simulatable.

We now want to compute the probability that Step 11 will not result in a restart. The probability that $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ can be computed by considering each coefficient separately. For each coefficient σ of $c\mathbf{s}_1$, the corresponding coefficient of $\mathbf{z}$ will be between $-\gamma_1 + \beta + 1$ and $\gamma_1 - \beta - 1$ (inclusively) whenever the corresponding coefficient of $\mathbf{y}_i$ is between $-\gamma_1 + \beta + 1 - \sigma$ and $\gamma_1 - \beta - 1 - \sigma$. The size of this range is $2(\gamma_1 - \beta) - 1$, and the coefficients of $\mathbf{y}$ have $2\gamma_1 - 1$ possibilities. Thus the probability that every coefficient of $\mathbf{y}$ is in the good range is

$$\left(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1 - 1}\right)^{\ell n} = \left(1 - \frac{\beta}{\gamma_1 - 1/2}\right)^{\ell n} \approx \mathrm{e}^{-n\beta\ell/\gamma_1}, \qquad (3)$$

where we used the fact that our values of γ_1 are large compared to $1/2$.

[6] With a more involved (and most likely messy) analysis, there is a good possibility that one could unconditionally prove that for any vector $\mathbf{r}$, $\Pr_{\mathbf{A},\mathbf{z},c}[\mathsf{HighBits}_q(\mathbf{Az} - c\mathbf{t}, \gamma_2) = r]$ is negligible.

We now move to computing the probability that we have

$$\|\mathbf{r}_0\|_\infty = \|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta\,.$$

If we (heuristically) assume that the low order bits are uniformly distributed modulo $2\gamma_2$, then there is a

$$\left(\frac{2(\gamma_2 - \beta) - 1}{2\gamma_2}\right)^{kn} \approx \mathrm{e}^{-n\beta k/\gamma_2}$$

probability that all the coefficients are in the good range (using the fact that our values of β are large compared to $1/2$.

As we already mentioned, if $\|c\mathbf{s}_2\|_\infty \le \beta$, then $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$ implies that $\mathbf{r}_1 = \mathbf{w}_1$. Thus the last check should succeed with overwhelming probability when the previous check passed. Therefore, the probability that Step 11 passes is

$$\approx \mathrm{e}^{-n\beta(\ell/\gamma_1 + k/\gamma_2)}\,. \qquad (4)$$

4.4 A Discussion on the Role of β.

From (4), we see that the smaller β is, the fewer repetitions will be needed to output a valid signature. On the other hand, we also saw in Section 4.3 that having $\|c\mathbf{s}_2\|_\infty \le \beta$ is necessary in order to perfectly simulate the distribution of a valid signer. A trivial bound for $\|c\mathbf{s}_i\|_\infty$ is $60 \cdot \eta$, which for the recommended parameter set in

Table 1 is 300. With such a value for β, the expected number of repetitions would go up from approximately 5 to 7.8 (using the formula in (4)), which is an increase of over 50% in the running time of the signing algorithm.

This is why we instead choose a smaller value for β such that $\Pr_{\mathbf{s},c}[\|c\mathbf{s}\|_\infty > \beta] \approx 2^{-80}$. If we assume that the signing algorithm will perform no more than 2^{64} signing operations, this implies that the signer will expect to encounter the scenario where $\|c\mathbf{s}_2\|_\infty > \beta$ less than once. If this scenario does arise, then the probability distribution (over the choice of $\mathbf{y}$) of the signature changes slightly in that it makes it impossible for some borderline values in the signature set $S^{\ell}_{\gamma_1-\beta-1}$ to appear. For example, if we encounter $\|c\mathbf{s}_2\|_\infty > \beta$ 10 out of 2^{64} times, then the expected number of times that the coefficient $\gamma_1-\beta-1$ will appear could go down from around $2^{64}/2^{20}$ to $2^{64}/2^{20} - 10$. Even if this slight decrease could be detected, it is unclear what the adversary would do with this information – he does not even know which of the signatures had the skewed samples. In this scenario, we remain very confident that the security of the scheme is not affected by the 2^{-80} probability of this bad event and believe that it is definitely not worthwhile to increase the value of β from 235 to 300.

Lowering β even further such that, for example, $\Pr_{\mathbf{s},c}[\|c\mathbf{s}\|_\infty > \beta] \approx 2^{-30}$ (making $\beta \approx 150$) is not advisable. An adversary who sees 2^{64} signatures would observe a noticeable skew. While we do not see a clear attack, such a discrepancy in probabilities looks, to us, a little worrisome. On the other hand, if the signature scheme were used in a scenario where only, say, 2^{20} signatures would ever be given out, then it could make sense to lower the β and decrease the running time by a factor of 2. We think that exploring the issues in this section more formally is a very interesting topic for further research.

4.5 Security Proof

Throughout the proof, we will make the assumption that the adversary gets $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ as the public key. This assumption is favorable to the adversary, as in the real scheme he only gets the high order bits of $\mathbf{t}$. In practice, therefore, the scheme may be even more difficult for the adversary to break.

Lemma 4.1. *Forging a signature implies finding* $\mathbf{u}_1, \mathbf{u}_2, u_3$ *such that* $\|\mathbf{u}_1\|_\infty \le 2\gamma_1, \|\mathbf{u}_2\|_\infty \le 4\gamma_2+2, \|u_3\|_\infty \le 2$ *such that* $\mathbf{A}\mathbf{u}_1+\mathbf{u}_2 = u_3\mathbf{t}_1 \cdot 2^d$ *and* $(\mathbf{u}_1, \mathbf{u}_2, u_3) \neq 0$. *Furthermore, vector* $\mathbf{u}_2$ *has at most* 2ω *coefficients of absolute value greater than* $2\gamma_2$.

Proof. The public key is set to be $pk = (\mathbf{A}, \mathbf{t})$. Signature queries by the adversary are created via the procedure described in Section 4.3. We now describe our extractor when the Adversary produces a winning query.

If the Adversary creates a valid signature $(\mathbf{z}, \mathbf{h}, c)$ for a message μ, then he must have queried

$$\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu) = c, \tag{5}$$

where $\mathbf{w}_1 = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$, either directly by querying H or indirectly during a signing query.

Case 1: If c was queried during a signing query, then the reduction already knows another $(\mathbf{z}', \mathbf{h}', c)$ (and the associated $\mathbf{w}_1'$) for a message μ' such that $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1', \mu') = c = \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$. This implies that $\mu = \mu'$ and $\mathbf{w}_1 = \mathbf{w}_1'$, or else we found a second pre-image for c. Since $\mathbf{w}_1 = \mathbf{w}_1'$, it must be that

$$\mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - \mathbf{t}_1 c \cdot 2^d, 2\gamma_2) = \mathbf{w}_1,$$
$$\mathsf{UseHint}_q(\mathbf{h}', \mathbf{A}\mathbf{z}' - \mathbf{t}_1 c \cdot 2^d, 2\gamma_2) = \mathbf{w}_1.$$

If $\mathbf{z} = \mathbf{z}'$, then Lemma 3.4 states that we must have $\mathbf{h} = \mathbf{h}'$, which is in contradiction with the assumption that the adversary has found a new signature. Thus it must be that $\mathbf{z} \neq \mathbf{z}'$. By Lemma 3.4, we also know that

$$\|\mathbf{A}\mathbf{z} - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2\|_\infty \le 2\gamma_2 + 1,$$
$$\|\mathbf{A}\mathbf{z}' - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2\|_\infty \le 2\gamma_2 + 1.$$

By the triangular inequality, this implies that

$$\|\mathbf{A}(\mathbf{z} - \mathbf{z}')\|_\infty \le 4\gamma_2 + 2\,.$$

This can be rewritten as

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') + \mathbf{u} = \mathbf{0}$$

for some $\mathbf{u}$ such that $\|\mathbf{u}\|_\infty \le 4\gamma_2 + 2$ where $\mathbf{z} - \mathbf{z}' \neq 0$. Because $\mathbf{h}$ and $\mathbf{h}'$ have all except ω elements equal to 0, we know from Lemma 3.4 that all but ω coefficients of $\mathbf{A}\mathbf{z} - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2$ and $\mathbf{A}\mathbf{z}' - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2$ are less than γ_2. Therefore all but 2ω coefficients of $\mathbf{u}$ are less than $2\gamma_2$.

Case 2: We now handle the case where the query in Equation (5) was done directly to H. A standard forking lemma argument shows that the reduction can then extract two signatures $(\mathbf{z}, \mathbf{h}, c)$ and $(\mathbf{z}', \mathbf{h}', c')$ for $c \neq c'$ such that

$$\mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - \mathbf{t}_1 c \cdot 2^d, 2\gamma_2) = \mathbf{w}_1,$$
$$\mathsf{UseHint}_q(\mathbf{h}', \mathbf{A}\mathbf{z}' - \mathbf{t}_1 c' \cdot 2^d, 2\gamma_2) = \mathbf{w}_1.$$

By Lemma 3.4, we know that

$$\|\mathbf{A}\mathbf{z} - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2\|_\infty \le 2\gamma_2 + 1,$$
$$\|\mathbf{A}\mathbf{z}' - \mathbf{t}_1 c' \cdot 2^d - \mathbf{w}_1 \cdot 2\gamma_2\|_\infty \le 2\gamma_2 + 1.$$

By the triangular inequality, this implies that

$$\|\mathbf{A}(\mathbf{z} - \mathbf{z}') - \mathbf{t}_1 \cdot 2^d \cdot (c - c')\|_\infty \le 4\gamma_2 + 2\,.$$

This can be rewritten as

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') + \mathbf{u} = \mathbf{t}_1 \cdot 2^d \cdot (c - c')$$

for some $\mathbf{u}$ such that $\|\mathbf{u}\|_\infty \le 4\gamma_2 + 2$. For the same reason as in the first case, at most 2ω coefficients of $\mathbf{u}$ can be greater than $2\gamma_2$. □

5 CONCRETE SECURITY ANALYSIS

We follow the general methodology from [3, 15] to analyze the security of our signature scheme, with minor adaptations. This methodology is significantly more conservative than prior ones used in lattice-based cryptography. In particular, we assume the adversary can run the asymptotically best algorithms known, with no overhead compared to the asymptotic run-times. In particular, we assume the adversary can cheaply handle huge amounts of (possibly quantum) memory.

We find this approach much sounder than relying on the currently best codes for the underlying tasks as the practical aspects of lattice algorithms have received little attention compared to integer factorization and discrete logarithm algorithms. Considering the gap between theory and practice of lattice reduction, and the

attention drawn to it due to potential deployment of lattice-based cryptography, we conclude that practical improvements are very likely to occur. This conservatism is in line with the goal of long-term post-quantum security. We note that despite this security analysis methodology, our schemes remain competitive in practice.

5.1 Lattice Reduction and Core-SVP Hardness

The best known algorithm for finding very short non-zero vectors in Euclidean lattices is the Block–Korkine–Zolotarev algorithm (BKZ) [50], proposed by Schnorr and Euchner in 1991. More recently, it was proven to quickly converge to its fix-point [32] and improved in practice [21]. Yet, what it achieves asymptotically remains unchallenged.

BKZ with block-size b makes calls to an algorithm that solves the Shortest lattice Vector Problem (SVP) in dimension b. The security of our scheme relies on the necessity to run BKZ with a large block-size b and the fact that the cost of solving SVP is exponential in b. The best known classical SVP solver [8] runs in time $\approx 2^{c_C \cdot b}$ with $c_C = \log_2 \sqrt{3/2} \approx 0.292$. The best known quantum SVP solver [34, Sec. 14.2.10] runs in time $\approx 2^{c_Q \cdot b}$ with $c_Q = \log_2 \sqrt{13/9} \approx 0.265$. One may hope to improve these run-times, but going below $\approx 2^{c_P \cdot b}$ with $c_P = \log_2 \sqrt{4/3} \approx 0.2075$ would require a theoretical breakthrough. Indeed, the best known SVP solvers rely on covering the b-dimensional sphere with cones of center-to-edge angle $\pi/3$: this requires $2^{c_P \cdot b}$ cones. The subscripts C, Q, P respectively stand for Classical, Quantum and Paranoid.

The strength of BKZ increases with b. More concretely, given as input a basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of an n-dimensional lattice, BKZ repeatedly uses the b-dimensional SVP-solver on lattices of the form $(\mathbf{c}_{i+1}(i), \ldots, \mathbf{c}_j(i))$ where $i \leq n$, $j = \min(n, i+b)$ and where $\mathbf{c}_k(i)$ denotes the projection of $\mathbf{c}_k$ orthogonally to the vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_i)$. The effect of these calls is to flatten the curve of the $\ell_i = \log_2 \|\mathbf{c}_i(i-1)\|$'s (for $i = 1, \ldots, n$). At the start of the execution, the ℓ_i's typically decrease fast, at least locally. As BKZ preserves the determinant of the $\mathbf{c}_i$'s, the sum of the ℓ_i's remains constant throughout the execution, and after a (small) polynomial number of SVP calls, BKZ has made the ℓ_i's decrease less. It can be heuristically estimated that for sufficiently large b, the local slope of the ℓ_i's converges to

$$slope(b) = \frac{1}{b-1} \log_2 \left(\frac{b}{2\pi \mathrm{e}} (\pi \cdot b)^{1/b} \right),$$

unless the local input ℓ_i's are already too small or too large. The quantity $slope(b)$ decreases with b, implying that the larger b the flatter the output ℓ_i's.

In our case, the input ℓ_i's are of the following form (cf. Fig. 2). The first ones are all equal to $\log_2 q$ and the last ones are all equal to 0. BKZ will flatten the jump, decreasing ℓ_i's with small i's and increasing ℓ_i's with large i's. However, the local slope $slope(b)$ may not be sufficiently small to make the very first ℓ_i's decrease and the very last ℓ_i's increase. Indeed, BKZ will not increase (resp. increase) some ℓ_i's if these are already smaller (resp. larger) than ensured by the local slope guarantee. In our case, the ℓ_i's are always of the following form at the end of the execution:

- The first ℓ_i's are constant equal to $\log_2 q$ (this is the possibly empty Zone 1).
- Then they decrease linearly, with slope $slope(b)$ (this is the never-empty Zone 2).
- The last ℓ_i's are constant equal to 0 (this is the possibly empty Zone 3).

The graph is continuous, i.e., if Zone 1 (resp. Zone 3) is not empty, then Zone 2 starts with $\ell_i = \log_2 q$ (resp. ends with $\ell_i = 0$).

5.2 Key-Recovery Attack: Solving Module-LWE

The attacker may attempt to recover the secret key $(\mathbf{s}_1, \mathbf{s}_2) \in R^\ell \times R^k$ from the public key $\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. As each of the $k + \ell$ elements of the secret key is sampled from S_η, this is exactly an instance of the Module-LWE problem.

Any such Module-LWE instance with dimensions ℓ, k can be viewed as an LWE instance of dimensions $256 \cdot \ell$ and $256 \cdot k$. Indeed, the above can be rewritten as finding $\mathrm{vec}(\mathbf{s}_1), \mathrm{vec}(\mathbf{s}_2) \in \mathbb{Z}^{256 \cdot \ell} \times \mathbb{Z}^{256 \cdot k}$ from $(\mathrm{rot}(\mathbf{A}), \mathrm{vec}(\mathbf{t}))$, where $\mathrm{vec}(\cdot)$ maps a vector of ring elements to the vector obtained by concatenating the coefficients of its coordinates, and $\mathrm{rot}(\mathbf{A}) \in \mathbb{Z}_q^{256 \cdot k \times 256 \cdot \ell}$ is obtained by replacing all entries $a_{ij} \in R_q$ of $\mathbf{A}$ by the 256×256 matrix whose z-th column is $\mathrm{vec}\left(x^{z-1} \cdot a_{ij}\right)$.

Given an LWE instance, there are two lattice-based attacks. The primal attack and the dual attack. Here, the primal attack consists in finding a short non-zero vector in the lattice $\Lambda = \{\mathbf{x} \in \mathbb{Z}^d : \mathbf{M}\mathbf{x} = \mathbf{0} \bmod \mathbf{q}\}$ where $\mathbf{M} = (\mathrm{rot}(\mathbf{A})_{[1:m]} | \mathbf{I}_m | \mathrm{vec}(\mathbf{t})_{[1:m]})$ is an $m \times d$ matrix where $d = 256 \cdot \ell + m + 1$ and $m \leq 256 \cdot k$. Indeed, it is sometime not optimal to use all the given equations in lattice attacks.

We tried all possible number m of rows, and, for each trial, we increased the blocksize of b until the value ℓ_{d-b} obtained as explained above was deemed sufficiently large. As explained in [3, Sec. 6.3], if $2^{\ell_{d-b}}$ is greater than the expected norm of $(\mathrm{vec}(\mathbf{s}_1), \mathrm{vec}(\mathbf{s}_2))$ after projection orthogonally to the first $d-b$ vectors, it is likely that the Module-LWE solution can be easily extracted from the BKZ output.

The dual attack consists in finding a short non-zero vector in the lattice $\Lambda' = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^d : \mathbf{M}^T\mathbf{x} + \mathbf{y} = 0 \bmod q)\}$, $\mathbf{M} = (\mathrm{rot}(\mathbf{A})_{[1:m]})$ is an $m \times d$ matrix where $d = 256 \cdot \ell$ and $m \leq 256 \cdot k$. Again, for each value of m, we increased the value of b until the value ℓ_1 obtained as explained above was deemed sufficiently small according the analysis of [3, Sec. 6.3].

5.3 Forgery Attack: Solving Module-SIS

The attacker may also attempt to forge a signature. By Lemma 4.1, this implies finding $\mathbf{u}_1, \mathbf{u}_2, u_3$ not all zero such that $\|\mathbf{u}_1\|_\infty \leq 2\gamma_1$, $\|\mathbf{u}_2\|_\infty \leq 4\gamma_2 + 2$, and $\|u_3\|_\infty \leq 2$ such that $\mathbf{A}\mathbf{u}_1 + \mathbf{u}_2 = u_3\mathbf{t}_1 \cdot 2^d$. This amounts to solving homogeneous Module-SIS for the matrix[7] $(\mathbf{A}|\mathbf{I}_k|\mathbf{t}_1)$ and infinity norm bounds $B = \max(2\gamma_1, 4\gamma_2 + 2, 2^{d+1})$. Note that the Module-SIS instance can be mapped to a SIS instance by considering the matrix $\mathrm{rot}(\mathbf{A}|\mathbf{I}_k|\mathbf{t}_1) \in \mathbb{Z}_q^{256 \cdot k \times 256 \cdot (\ell + k + 1)}$. The attacker may consider a subset of w columns, and let the solution coefficients corresponding to the dismissed columns be zero.

Remark 5.1. An unusual aspect here is that we are considering the infinity norm, rather than the Euclidean norm. Further, for

[7] One could tweak this matrix to $(\mathbf{A}|\mathbf{I}_k|\mathbf{t}_1 2^d)$ to enforce stronger bounds ≤ 2 on the last coefficients, but it severely complicates the analysis. This approximation is made in favor of the adversary, whose real task is harder than what we analyze: this provides a lower bound on the cost of such an attack.

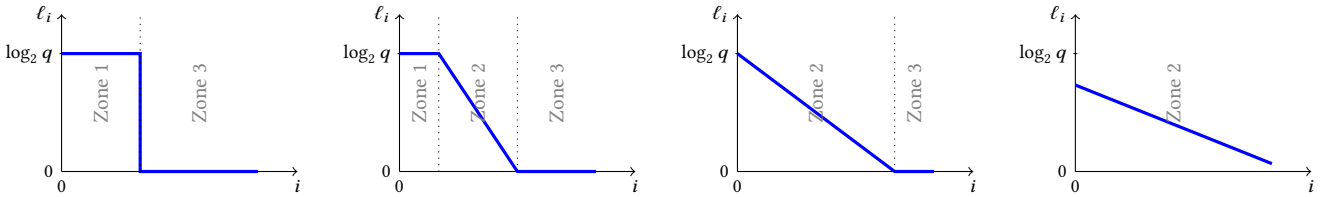

Figure 2: Evolution of Gram-Schmidt length in log-scale under BKZ reduction for various blocksizes. The area under the curves remains constant, and the slope in Zone 2 decrease with the blocksize. Note that Zone 3 may disappear before Zone 1, depending on the shape of the input basis.

our specific parameters, the Euclidean norms of the solutions are above q. In particular, the vector $(q, 0, \ldots, 0)^T$ belongs to the lattice, has Euclidean norm below that of the solution, but its infinity norm above the requirement. This raises difficulties in analyzing the strength of BKZ towards solving our infinity norm SIS instances: indeed, even with small values of b, the first ℓ_i's are short (they correspond to q-vectors), even though they are not solutions.

For each number w of selected columns and for each value of b, we compute the estimated BKZ output ℓ_i's, as explained above. We then consider the smallest i such that ℓ_i is below $\log_2 q$ and the largest j such that ℓ_j above 0. These correspond to the vectors that were modified by BKZ, with smallest and largest indices, respectively. In fact, for the same cost as a call to the SVP-solver, we can obtain $\sqrt{4/3}^b$ vectors with Euclidean norm $\approx 2^{\ell_i}$ after projection orthogonally to the first $i-1$ basis vectors. Now, let us look closely at the shape of such a vector. As the first $i-1$ basis vectors are the first $i-1$ canonical unit vectors multiplied by q, projecting orthogonally to these consists in zeroing the first $i-1$ coordinates. The remaining $w-i+1$ coordinates have total Euclidean norm $\approx 2^{\ell_i} \approx q$, and the last $w-j$ coordinates are 0. We heuristically assume that these coordinates have similar magnitudes $\sigma \approx 2^{\ell_i}/\sqrt{j-i+1}$; we model each such coordinate as a Gaussian of standard deviation σ. We assume that each one of our $\sqrt{4/3}^b$ vectors has its first $i-1$ coordinates independently uniformly distributed modulo q, and finally compute the probability that all coordinates in both ranges $[0, i-1]$ and $[i, j]$ are less than B in absolute value. Our cost estimate is the inverse of that probability multiplied by the run-time of our b-dimensional SVP-solver.

Forgetting q-vectors. For all the parameter sets proposed in this paper, the best parametrization of the attack above kept the basis in a shape with a non-trivial Zone 1. We note that the coordinates in this range have a quite lower probability of passing the ℓ_∞ constraint than coordinates in Zone 2. We therefore considered a strategy consisting of "forgetting" the q-vectors, by re-randomizing the input basis before running the BKZ algorithm. For the same blocksize b, this makes Zone 1 of the output basis disappear (BKZ does not find the q-vectors), at the cost of producing a basis with first vectors of larger Euclidean norms. This is depicted in Fig. 3.

It turns out that this strategy always improves over the previous strategy for the parameter ranges considered in this paper. We therefore used this strategy for our security estimates.

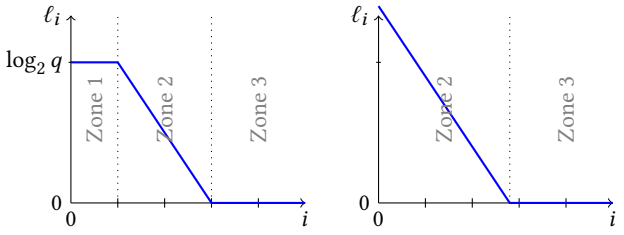

Figure 3: Effect of forgetting q-vectors by randomization, under the same BKZ-blocksize b.

5.4 On Other Attacks

For our parameters, the BKW [14] and Arora–Ge [5] families of algorithms are far from competitive.

Algebraic attacks. One specificity of our LWE and SIS instances is that they are inherited from Module-LWE and Module-SIS instances. One may wonder whether the extra algebraic structure of the resulting lattices can be exploited by an attacker. The line of work of [12, 18, 22, 23] did indeed found new cryptanalytic results on certain algebraic lattices, but [23] mentions serious obstacles towards breaking cryptographic instances of Ring-LWE. By switching from Ring-LWE to Module-LWE, we get even further away from those weak algebraic lattice problems.

Dense sublattice attacks. Kirchner and Fouque [33] showed that the existence of many linearly independent and unexpectedly short lattice vectors (much shorter than Minkowski's bound) helps BKZ run better than expected in some cases. This could happen for our primal LWE attack, by extending $\mathbf{M} = (\mathrm{rot}(\mathbf{A})_{[1:\mathbf{m}]}|\mathbf{I_m}|\mathrm{vec}(\mathbf{t})_{[1:\mathbf{m}]})$ to $(\mathrm{rot}(\mathbf{A})_{[1:m]}|\mathbf{I}_m|\mathrm{rot}(\mathbf{t})_{[1:m]})$: the associated lattice now has 256 linearly independent short vectors rather than a single one. The Kirchner-Fouque analysis of BKZ works best if both q and the ratio between the number of unexpectedly short vectors and the lattice dimension are high. In the NTRU case, for example, the ratio is $1/2$, and, for some schemes derived from NTRU, the modulus q is also large. We considered this refined analysis of BKZ in our setup, but, to become relevant for our parameters, it requires a parameter b which is higher than needed with the usual analysis of BKZ. Note that [33] also arrived to the conclusion that this attack is irrelevant in the small modulus regime, and is mostly a

threat to fully homomorphic encryption schemes and cryptographic multilinear maps.

Note that, once again, the switch from Ring-LWE to Module-LWE takes us further away from lattices admitting unconventional attacks. Indeed, the dimension ratio of the dense sub-lattice is $1/2$ in NTRU, at most $1/3$ in lattices derived from Ring-LWE, and at most $1/(\ell+2)$ in lattices derived from Module-LWE.

Specialized attack against ℓ_∞-SIS. At last, we would like to mention that it is not clear whether the attack sketched in Section 5.3 above for SIS in infinity norm is optimal. Indeed, as we have seen, this approach produces many vectors, with some rather large uniform coordinates (at indices $1, \ldots, i$), and smaller Gaussian ones (at indices $i, \ldots, j$). In our current analysis, we simply hope that one of the vector satisfies the ℓ_∞ bound. Instead, one could combine them in ways that decrease the size of the first (large) coefficients, while letting the other (small) coefficients grow a little bit.

This situation created by the use of ℓ_∞-SIS (see Remark 5.1) has — to the best of our knowledge — not been studied in detail. After a preliminary analysis, we do not consider such an improved attack a serious threat to our concrete security claims, especially in the light of the approximations already made in the favor of the adversary. Nevertheless, we believe this question deserves a detailed study, which we leave to future work.

6 IMPLEMENTATION

To illustrate the performance of the Dilithium signature scheme we implemented the scheme with our parameter sets, using optimizations in Intel's AVX2 vector instruction set. We benchmark the implementation on one core of an Intel Core-i7 4770k (Haswell) processor. We follow the standard practice of disabling hyperthreading and TurboBoost.

6.1 Primitives and Reference Implementation

In previous sections, Dilithium was introduced in abstract terms without fixing concrete instantiations of the function Sam. This subsection provides concrete detail on our implementation choices.

Symmetric primitives. We decided to instantiate Sam with the expandable output function SHAKE-128, standardized in FIPS 202 [45] so that Dilithium rely on the Keccak-f1600 permutation, standardized after years of cryptanalytic scrutiny through the course of the SHA-3 competition. Obviously, different implementations are free to use whichever pseudo-random generator is offering the best performance and security on their respective platform. Other options include the ChaCha20 stream cipher [10], the BLAKE2X extendable output function [6], or SHA256 for all hashes (with "output extension for G via MGF1"; see [44, App. B.2.1]), and AES in counter mode for the expansion of seeds; this latter choice would certainly be faster than SHAKE-128 on platforms with hardware AES and SHA256 support.

NTT computations. For polynomial multiplication we use an NTT-based algorithm as is standard in lattice-based cryptography (see [3, 25, 31, 39, 42, 48, 49], among others). Contrary to previous implementations that utilized floating point vector instructions (see, for example, [3, 31]), our vectorized AVX2-code is written with integer instructions only. For fast constant-time reductions modulo q we use the Montgomery algorithm, where floating point implementations typically multiply by a precomputed floating point inverse of q and round. We make heavy use of lazy reductions and only reduce values when they do not fit into 32 bit or a standard representative is needed. A full multiplication of two polynomials in the ring $\mathbb{Z}_q[X]/(X^{256}+1)$ comprising two forward NTTs, one inverse NTT, and pointwise multiplication, requires about $5,000$ cycles.

Generation of noise. Noise polynomials in Dilithium have coefficients sampled uniformly in $[-\eta, \eta]$. To obtain such a noise polynomial, we first sample a random 32-byte seed r and expand it using SHAKE-128 (i.e., $\mathsf{Sam}(r)$). We then use rejection sampling on each output byte. (As stated above, the choice of how the 256 uniformly random bytes are generated can be a local, platform-dependent choice.)

6.2 Results and Comparisons with Related Works

Our results are presented in Table 1. In particular, for our recommended parameter set, we measure a total of $1,042,250$ cycles for the complete signature generation. Key generation costs $251,590$ cycles, and verification consumes $297,590$ cycles. Note that our "very high" parameter set achieves better performance for a larger signature due to the fact that the manner in which the parameters were set results in a smaller rejection rate.

In order to compare these results to other post-quantum signature schemes, we consider the performance figures reported in [11, 20]. (We note that BLISS [25] did not claim any post-quantum security.) The MQDSS signature generation is reported to take 8 510 616 cycles on an Intel Core i7-4770K CPU at 3.5GHz (with an "extensive use of AVX2 instructions") [20] for a signature of 40 952 Bytes. The hash-based SPHINCS signature generation is reported to take 51 636 372 cycles on an Intel Core i7-4770K CPU at 3.5GHz (with "focused [...] optimization efforts on [the signing procedure]") for a signature of $\approx$ 41 000 Bytes [11]. These results provide confidence that our implementation, using AVX2 instructions, is performing very favourably.

ACKNOWLEDGMENTS

Léo Ducas was supported by a Veni Innovational Research Grant from NWO under project number 639.021.645. Vadim Lyubashevsky and Gregor Seiler were supported by the SNSF ERC Transfer Starting Grant CRETP2-166734-FELICITY and the H2020 Project SAFEcrypto. Peter Schwabe was supported by ICT Programme 115 under contract ICT-645622 PQCRYPTO. Damien Stehlé was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC.

REFERENCES

[1] Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. 2012. Tightly-Secure Signatures from Lossy Identification Schemes. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 572–590.

[2] Erdem Alkim, Nina Bindel, Johannes Buchmann, Ozgur Dagdelen, Edward Eaton, Gus Gutoski, Juliane Kramer, and Filip Pawlega. 2015. Revisiting TESLA in the quantum random oracle model. Cryptology ePrint Archive, Report 2015/755. (2015). http://eprint.iacr.org/2015/755.

[3] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum key exchange – a new hope. In *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 327–343. http://cryptojedi.org/papers/#newhope.

[4] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. 2014. Quantum Attacks on Classical Proof Systems: The Hardness of Quantum Rewinding. In *55th FOCS*. IEEE Computer Society Press, 474–483. https://doi.org/10.1109/FOCS.2014.57

[5] Sanjeev Arora and Rong Ge. 2011. New Algorithms for Learning in Presence of Errors. In *ICALP 2011, Part I (LNCS)*, Luca Aceto, Monika Henzinger, and Jiri Sgall (Eds.), Vol. 6755. Springer, Heidelberg, 403–415.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2016. BLAKE2X. (2016). https://blake2.net/blake2x.pdf.

[7] Shi Bai and Steven D. Galbraith. 2014. An Improved Compression Technique for Signatures Based on Learning with Errors. In *CT-RSA 2014 (LNCS)*, Josh Benaloh (Ed.), Vol. 8366. Springer, Heidelberg, 28–47. https://doi.org/10.1007/978-3-319-04852-9_2

[8] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. 2016. New directions in nearest neighbor searching with applications to lattice sieving. In *27th SODA*, Robert Krauthgamer (Ed.). ACM-SIAM, 10–24. https://doi.org/10.1137/1.9781611974331.ch2

[9] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, V. Ashby (Ed.). ACM Press, 62–73.

[10] Daniel J. Bernstein. 2008. ChaCha, a variant of Salsa20. In *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*. http://cr.yp.to/papers.html#chacha.

[11] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: Practical Stateless Hash-Based Signatures. In *EUROCRYPT 2015, Part I (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9056. Springer, Heidelberg, 368–397. https://doi.org/10.1007/978-3-662-46800-5_15

[12] Jean-François Biasse and Fang Song. 2016. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *27th SODA*, Robert Krauthgamer (Ed.). ACM-SIAM, 893–902. https://doi.org/10.1137/1.9781611974331.ch64

[13] Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. 2013. Why "Fiat-Shamir for Proofs" Lacks a Proof. In *TCC*. 182–201.

[14] Avrim Blum, Adam Kalai, and Hal Wasserman. 2003. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM* 50, 4 (2003), 506–519.

[15] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In *ACM CCS 16*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1006–1018.

[16] Xavier Boyen. 2010. Lattice Mixing and Vanishing Trapdoors: A Framework for Fully Secure Short Signatures and More. In *PKC 2010 (LNCS)*, Phong Q. Nguyen and David Pointcheval (Eds.), Vol. 6056. Springer, Heidelberg, 499–517.

[17] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. 2016. Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme. In *CHES 2016 (LNCS)*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.), Vol. 9813. Springer, Heidelberg, 323–345. https://doi.org/10.1007/978-3-662-53140-2_16

[18] Peter Campbell, Michael Groves, and Dan Shepherd. 2014. Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*. 1–9.

[19] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. 2016. Report on Post-Quantum Cryptography. NISTIR 8105. (2016). http://dx.doi.org/10.6028/NIST.IR.8105.

[20] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. 2016. From 5-Pass MQ-Based Identification to MQ-Based Signatures. In *ASIACRYPT 2016, Part II (LNCS)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10032. Springer, Heidelberg, 135–165. https://doi.org/10.1007/978-3-662-53890-6_5

[21] Yuanmi Chen and Phong Q. Nguyen. 2011. BKZ 2.0: Better Lattice Security Estimates. In *ASIACRYPT 2011 (LNCS)*, Dong Hoon Lee and Xiaoyun Wang (Eds.), Vol. 7073. Springer, Heidelberg, 1–20.

[22] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. 2016. Recovering Short Generators of Principal Ideals in Cyclotomic Rings. In *EUROCRYPT 2016, Part II (LNCS)*, Marc Fischlin and Jean-Sébastien Coron (Eds.), Vol. 9666. Springer, Heidelberg, 559–585. https://doi.org/10.1007/978-3-662-49896-5_20

[23] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. 2017. Short Stickelberger Class Relations and Application to Ideal-SVP. In *EUROCRYPT (1) (Lecture Notes in Computer Science)*, Vol. 10210. 324–348.

[24] Ivan Damgård, Serge Fehr, and Louis Salvail. 2004. Zero-Knowledge Proofs and String Commitments Withstanding Quantum Attacks. In *CRYPTO 2004 (LNCS)*, Matthew Franklin (Ed.), Vol. 3152. Springer, Heidelberg, 254–272.

[25] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice Signatures and Bimodal Gaussians. In *CRYPTO 2013, Part I (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8042. Springer, Heidelberg, 40–56. https://doi.org/10.1007/978-3-642-40041-4_3

[26] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. 2014. Efficient Identity-Based Encryption over NTRU Lattices. In *ASIACRYPT 2014, Part II (LNCS)*, Palash Sarkar and Tetsu Iwata (Eds.), Vol. 8874. Springer, Heidelberg, 22–41. https://doi.org/10.1007/978-3-662-45608-8_2

[27] Léo Ducas and Daniele Micciancio. 2014. Improved Short Lattice Signatures in the Standard Model. In *CRYPTO 2014, Part I (LNCS)*, Juan A. Garay and Rosario Gennaro (Eds.), Vol. 8616. Springer, Heidelberg, 335–352. https://doi.org/10.1007/978-3-662-44371-2_19

[28] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, 197–206.

[29] Shafi Goldwasser and Yael Tauman Kalai. 2003. On the (In)security of the Fiat-Shamir Paradigm. In *44th FOCS*. IEEE Computer Society Press, 102–115.

[30] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2012. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In *CHES*. 530–547.

[31] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. 2013. Software Speed Records for Lattice-Based Signatures. In *PQCrypto (Lecture Notes in Computer Science)*, Vol. 7932. Springer, 67–82.

[32] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. 2011. Analyzing Blockwise Lattice Algorithms Using Dynamical Systems. In *CRYPTO 2011 (LNCS)*, Phillip Rogaway (Ed.), Vol. 6841. Springer, Heidelberg, 447–464.

[33] Paul Kirchner and Pierre-Alain Fouque. 2017. Revisiting Lattice Attacks on Overstretched NTRU Parameters. In *EUROCRYPT (1) (Lecture Notes in Computer Science)*, Vol. 10210. 3–26.

[34] Thijs Laarhoven. 2015. *Search problems in cryptography*. Ph.D. Dissertation. Eindhoven University of Technology.

[35] Adeline Langlois and Damien Stehlé. 2015. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography* 75, 3 (2015), 565–599. https://doi.org/10.1007/s10623-014-9938-4

[36] Vadim Lyubashevsky. 2009. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *ASIACRYPT 2009 (LNCS)*, Mitsuru Matsui (Ed.), Vol. 5912. Springer, Heidelberg, 598–616.

[37] Vadim Lyubashevsky. 2012. Lattice Signatures without Trapdoors. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Heidelberg, 738–755.

[38] Vadim Lyubashevsky and Daniele Micciancio. 2006. Generalized Compact Knapsacks Are Collision Resistant. In *ICALP 2006, Part II (LNCS)*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.), Vol. 4052. Springer, Heidelberg, 144–155.

[39] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. 2008. SWIFFT: A Modest Proposal for FFT Hashing. In *FSE 2008 (LNCS)*, Kaisa Nyberg (Ed.), Vol. 5086. Springer, Heidelberg, 54–72.

[40] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010 (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Heidelberg, 1–23.

[41] Vadim Lyubashevsky and Daniel Wichs. 2015. Simple Lattice Trapdoor Sampling from a Broad Class of Distributions. In *PKC 2015 (LNCS)*, Jonathan Katz (Ed.), Vol. 9020. Springer, Heidelberg, 716–730. https://doi.org/10.1007/978-3-662-46447-2_32

[42] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. 2016. NFLlib: NTT-Based Fast Lattice Library. In *CT-RSA 2016 (LNCS)*, Kazue Sako (Ed.), Vol. 9610. Springer, Heidelberg, 341–356. https://doi.org/10.1007/978-3-319-29485-8_20

[43] Daniele Micciancio and Michael Walter. 2017. Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time. *IACR Cryptology ePrint Archive* 2017 (2017), 259. http://eprint.iacr.org/2017/259

[44] Kathleen M. Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. 2016. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. (2016). https://tools.ietf.org/html/rfc8017.

[45] NIST. 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Technical Report. Available at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`.

[46] Chris Peikert and Alon Rosen. 2006. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In *TCC 2006 (LNCS)*, Shai Halevi and Tal Rabin (Eds.), Vol. 3876. Springer, Heidelberg, 145–166.

[47] Peter Pessl. 2016. Analyzing the Shuffling Side-Channel Countermeasure for Lattice-Based Signatures. In *INDOCRYPT 2016 (LNCS)*, Orr Dunkelman and Somitra Kumar Sanadhya (Eds.), Vol. 10095. Springer, Heidelberg, 153–170. https://doi.org/10.1007/978-3-319-49890-4_9

[48] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. 2014. Enhanced Lattice-Based Signatures on Reconfigurable Hardware. In *CHES 2014 (LNCS)*, Lejla Batina

and Matthew Robshaw (Eds.), Vol. 8731. Springer, Heidelberg, 353–370. https://doi.org/10.1007/978-3-662-44709-3_20
[49] Thomas Pöppelmann and Tim Güneysu. 2012. Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware. In *LATINCRYPT 2012 (LNCS)*, Alejandro Hevia and Gregory Neven (Eds.), Vol. 7533. Springer, Heidelberg, 139–158.
[50] Claus-Peter Schnorr and M. Euchner. 1994. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.* 66 (1994), 181–199.
[51] Dominique Unruh. 2017. Post-Quantum Security of Fiat-Shamir. *IACR Cryptology ePrint Archive* 2017 (2017), 398. http://eprint.iacr.org/2017/398

A PRELIMINARIES FOR DILITHIUM-G

We use the same rings as in Dilithium. In particular, we work over R and R_q with the same $n = 256$ and $q = 8380417$. The main implementation difference between Dilithium and Dilithium-G is that in the signing algorithm of the latter we will be performing sampling from a discrete Gaussian distribution rather than from the uniform one.

Discrete Gaussian distribution. Since the ring R is isomorphic to $\mathbb{Z}^{256}$ as an additive group and has the same geometry, rather than sampling over $\mathbb{Z}^{256}$, we will say that we are sampling over R. Similarly, the set R^k has the same geometry as $\mathbb{Z}^{256\cdot k}$. For $\sigma \in \mathbb{R}^+$ and positive integer k, we define the distribution over R^k,

$$D^k_\sigma(\mathbf{r}) = \rho_\sigma(\mathbf{r})/\rho_\sigma(R^k) \quad \text{where} \quad \rho_\sigma(R^k) = \sum_{\mathbf{r}\in R^k} \rho_\sigma(\mathbf{r}),$$

and $\rho_\sigma(\mathbf{r}) = \exp(-\|\mathbf{r}\|^2/(2\sigma^2))$ for $\mathbf{r} \in R^k$.

Producing "hints". We introduce the procedures $\mathsf{MakeGHint}_q$ and $\mathsf{UseGHint}_q$ for generating and using hints, respectively (see Algorithms 11 and 12), that will be used in Dilithium-G instead of $\mathsf{MakeHint}_q$ and $\mathsf{UseHint}_q$.

Algorithm 11 $\mathsf{MakeGHint}_q(z, r, \alpha)$

$m := (q-1)/\alpha$
$r_1 := \mathsf{HighBits}_q(r, \alpha)$
$v_1 := \mathsf{HighBits}_q(r+z, \alpha)$
return $(v_1 - r_1) \bmod^{\pm} m$

Algorithm 12 $\mathsf{UseGHint}_q(y, r, \alpha)$

$m := (q-1)/\alpha$
$r_1 := \mathsf{HighBits}_q(r, \alpha)$
return $(r_1 + y) \bmod^{+} m$

The main difference with $\mathsf{MakeHint}_q$ and $\mathsf{UseHint}_q$ from Section 3 is that the hints are no longer restricted to being one bit. The reason is that the small integer z is no longer bounded within an interval, but is rather generated according to a Gaussian distribution. It would be very wasteful to our scheme to pick an interval that is large enough so that the carry vector can be at most 1. For this reason, we allow z to cause larger carries.

Lemmas A.1 and A.2 are analogous to Lemmas 3.1 and 3.3 from Section 3.2. We will not need an equivalent of Lemma 3.2 because the verification procedure will implicitly check the length of the solution.

Lemma A.1. *Let r, z be integers. Then*

$$\mathsf{UseGHint}_q(\mathsf{MakeGHint}_q(z, r, \alpha), r, \alpha) = \mathsf{HighBits}_q(z + r, \alpha).$$

Proof. Let $v_1 = \mathsf{HighBits}_q(r + z, \alpha)$. We have that

$$\mathsf{UseGHint}_q\left(\mathsf{MakeGHint}_q(z, r, \alpha), r, \alpha\right) = v_1 \bmod^{+} m.$$

Thus we just need to show that v_1 is always a value between 0 and $m-1$. In other words, we need to show that the $\mathsf{Decompose}_q$ routine always outputs (r_1, r_0) where $0 \le r_1 < m-1$. Note that $r_1 = (r - (r \bmod^{\pm} \alpha))/\alpha$. The numerator is always between 0 and $q-1$ (inclusively), but the procedure sets it to 0 if it is $q-1$. We thus have $0 \le r_1 < (q-1)/\alpha = m$. □

Lemma A.2. *Let $m = (q-1)/\alpha$, $r \in \mathbb{Z}_q$ and $y, y' \in (-m/2, m/2]$ integers. If $\mathsf{UseGHint}_q(y, r, \alpha) = \mathsf{UseGHint}_q(y', r, \alpha)$, then $y = y'$.*

Proof. If $\mathsf{UseGHint}_q(y, r, \alpha) = \mathsf{UseGHint}_q(y', r, \alpha)$, then $y = y' \bmod^{+} m$. Since $|y|, |y'| \le m/2$, this is only possible if $y = y'$. □

As in Section 3.2, we deduce the following lemma.

Lemma A.3. *Suppose that q and α are positive integers satisfying $q \equiv 1 \pmod{\alpha}$. Let $m = (q-1)/\alpha$, $\mathbf{r}$ and $\mathbf{z}$ be vectors of elements in R_q and $\mathbf{y}, \mathbf{y}'$ be integral vectors of elements in $(-m/2, m/2]$. Then the $\mathsf{Decompose}_q$, $\mathsf{MakeGHint}_q$, and $\mathsf{UseGHint}_q$ algorithms satisfy the following properties:*

(1) $\mathsf{UseGHint}_q(\mathsf{MakeGHint}_q(\mathbf{z}, \mathbf{r}, \alpha), \mathbf{r}, \alpha) = \mathsf{HighBits}_q(\mathbf{z} + \mathbf{r}, \alpha)$.
(2) *If* $\mathsf{UseGHint}_q(\mathbf{y}, \mathbf{r}, \alpha) = \mathsf{UseGHint}_q(\mathbf{y}', \mathbf{r}, \alpha)$, *then* $\mathbf{y} = \mathbf{y}'$.

B DILITHIUM-G: NORMALLY-DISTRIBUTED DIGITAL SIGNATURES

In this section, we propose a variant of Dilithium, called Dilithium-G, that uses a discrete Gaussian distribution instead of the uniform distribution S_η (Appendix A introduces the new notation used in this section). The main conceptual difference between this signature and Dilithium is in the way that the rejection sampling is done. In Dilithium, the rejection sampling is used to make the distribution of $\mathbf{z}$ uniform in a hypercube, whereas Dilithium-G uses a different kind of rejection sampling which makes the vector $(\mathbf{z}_1, \mathbf{z}_2)$ take the discrete Gaussian distribution. As was shown in [37], rejection sampling in this manner can create signatures that are asymptotically shorter (in the ℓ_2-norm) by a factor of $O(\sqrt{n \cdot (k+\ell)})$.

Other than the difference in the rejection sampling algorithm, there is also a slight difference in the output of the $\mathsf{MakeHint}_q$ algorithm. In the previous scheme, the smaller term in the $\mathsf{MakeHint}_q$ could only increase or decrease by 1 the higher order coefficient of the sum. When the coefficients are discrete Gaussians, however, there is no longer a hard bound, and so the higher order coefficients of the sum can change by other values. Therefore the hint $\mathbf{h}$ is no longer encoding 0's or 1's as before, but rather larger integers.

B.1 The Signature Scheme

Dilithium-G is described in Algorithms 13 to 15.

Algorithm 13 KeyGen()

1: $\rho' \leftarrow \{0,1\}^{256}$
2: $(\mathbf{s}_1, \mathbf{s}_2) \sim S_\eta^\ell \times S_\eta^k := \mathsf{Sam}(\rho')$
3: **if** max singular value of $\begin{bmatrix}\mathrm{rot}(\mathbf{s}_1)\\ \mathrm{rot}(\mathbf{s}_2)\end{bmatrix} \in \mathbb{Z}^{256(k+\ell)\times 256}$ is larger than S, **restart**
4: $\rho \leftarrow \{0,1\}^{256}$
5: $\mathbf{A} \sim R_q^{k\times\ell} := \mathsf{Sam}(\rho)$
6: $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
7: $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
8: **return** $(pk := (\mathbf{t}_1, \rho), sk := (\rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}))$

The key generation proceeds exactly as in Dilithium, except we now check to make sure that the largest singular value of the integer matrix $\begin{bmatrix}\mathrm{rot}(\mathbf{s}_1)\\ \mathrm{rot}(\mathbf{s}_2)\end{bmatrix} \in \mathbb{Z}^{256(k+\ell)\times 256}$, where rot$(\cdot)$ is defined as in Section 5.2,is not too large. The cut-off bound is chosen heuristically so that there is about a 50% chance that random $\mathbf{s}_1, \mathbf{s}_2$ satisfy the bound. The singular value bound will assure that $\|c(\mathbf{s}_1, \mathbf{s}_1)^T\|$ will not be too large.

Algorithm 14 Sign$(sk = (\rho, \mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2)^T, \mathbf{t}), \mu \in \mathcal{M})$

1: $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$
2: $\mathbf{t}_0 := \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
3: $r \leftarrow \{0,1\}^{256}$
4: $(\mathbf{y}_1, \mathbf{y}_2) \sim D_\sigma^\ell \times D_\sigma^k := \mathsf{Sam}(r)$
5: $\mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$
6: $(\mathbf{w}_1, \mathbf{w}_0) := \mathsf{Decompose}_q(\mathbf{w}, \alpha)$
7: $c \leftarrow \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$
8: $\mathbf{z} = \begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\end{bmatrix} := \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix} + c\begin{bmatrix}\mathbf{s}_1\\ \mathbf{s}_2\end{bmatrix}$
9: $u \leftarrow [0,1)$
10: **if** $u > (1/M) \cdot \exp((-2\langle \mathbf{z}, c\mathbf{s}\rangle + \|c\mathbf{s}\|^2)/(2\sigma^2))$ **then** goto 3
11: $\mathbf{z}_2 := \mathbf{z}_2 - c\mathbf{t}_0 - \mathbf{w}_0$
12: **if** $\|(\mathbf{z}_1, \mathbf{z}_2)\| \geq B$ **then** goto 3
13: $\mathbf{h} := \mathsf{MakeGHint}_q(\mathbf{z}_2, \alpha\mathbf{w}_1 - \mathbf{z}_2, \alpha)$
14: **return** $\Sigma := (\mathbf{z}_1, \mathbf{h}, c)$

Algorithm 15 Verify$(pk = (\rho, \mathbf{t}_1), \mu \in \mathcal{M}, \Sigma = (\mathbf{z}_1, \mathbf{h}, c))$

1: $\mathbf{w}_1 := \mathsf{UseGHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1 \cdot 2^d, \alpha)$
2: **if** $c = H(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ and $\|(\mathbf{z}_1, \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1 \cdot 2^d - \alpha\mathbf{w}_1)\| < B$ and $\|\mathbf{h}\|_\infty \leq (q-1)/2\alpha$ **then**
3: **return** 1
4: **else**
5: **return** 0
6: **end if**

The signing procedure begins by computing $\mathbf{t}_0$ in the same manner as in the previous algorithm. The signer then samples $\mathbf{y}_1, \mathbf{y}_2$ from a discrete Gaussian distribution with standard deviation σ (using an extendable output function Sam expanding a seed r), computes $\mathbf{w} = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$ and decomposes it into $\mathbf{w} = \alpha\mathbf{w}_1 + \mathbf{w}_0$. The polynomial c is then computed as in the previous signature scheme, and $\mathbf{z}_i$ is computed as $\mathbf{y}_i + c\mathbf{s}_i$. The signature $\mathbf{z}_1, \mathbf{z}_2$ is accepted with probability $(1/M) \cdot \exp(-2\langle \mathbf{z}, c\mathbf{s}\rangle + \|c\mathbf{s}\|^2/(2\sigma^2))$. In order for this to be a valid probability, one needs to set the constant M so that the value should always be less than 1. This constant M is also the expected number of repetitions that will be needed to output one signature. The precise relationship between M, σ, and $\|c\mathbf{s}\|$ was derived in [37] and we state it in Lemma B.1.

If this rejection sampling step passes, then the distribution of $\mathbf{z}$ is a discrete Gaussian centered at 0. With high probability, the ℓ_2 norm of this vector should be tightly concentrated around a value close to $\sigma \cdot \sqrt{(k+\ell)\cdot 256}$. Because we perturb this vector (due to the dropping of the lower order terms of $\mathbf{z}_2$ and $\mathbf{t}$), the norm increases slightly. Experimentally, we set the bound

$$B^2 = \left(1.05 \cdot \sigma\sqrt{(k+\ell)\cdot 256}\right)^2 + \left(2^{d-1} \cdot \sqrt{60 \cdot 256 \cdot k}\right)^2, \quad (6)$$

such that the ℓ_2 norm of the vector in Step 12 is almost always less than B. If this step passes, then the next step of the algorithm uses $\mathbf{z}_2$ and $\mathbf{w}_1$ to construct a hint $\mathbf{h}$ that allows one to derive $\mathbf{w}_1$ from $\alpha\mathbf{w}_1 - \mathbf{z}_2$ and this hint. The verification procedure also checks that every coefficient of $\mathbf{h}$ is smaller than $(q-1)/2\alpha$. We do this check so that Lemma A.2 can be applied to show strong unforgeability. Since α is close to the standard deviation σ and each coefficient of $\mathbf{h}$ essentially represents the number of standard deviations $\mathbf{z}_2$ was larger than σ, with very high probability (greater than $1 - 2^{-128}$), the magnitude of each coefficient of $\mathbf{h}$ is less than 15, thus this verification step will always pass for an honest signer.

Concrete parameters. We provide concrete parameters and security estimates in Table 2.

B.2 Correctness

First we show that the norm bound check in Step 12 is the same as in the verification algorithm.

$$\mathbf{z}_2 = \mathbf{y}_2 + c\mathbf{s}_2 - c\mathbf{t}_0 - \mathbf{w}_0 = \alpha\mathbf{w}_1 + \mathbf{w}_0 - \mathbf{A}\mathbf{y}_1 + c\mathbf{s}_2 - c\mathbf{t}_0 - \mathbf{w}_0$$
$$= \alpha\mathbf{w}_1 - \mathbf{A}\mathbf{z}_1 + c(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2) - c\mathbf{t}_0 = \alpha\mathbf{w}_1 - \mathbf{A}\mathbf{z}_1 + c\mathbf{t}_1 \cdot 2^d.$$

Then we want to show that the first step in the verification equation produces $\mathbf{w}_1$. From the above equality, we know that

$$\alpha\mathbf{w}_1 - \mathbf{z}_2 = \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1 \cdot 2^d, \quad (7)$$

and so Lemma A.3 gives us that $\mathbf{w}_1 = \mathsf{UseGHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1 \cdot 2^d, \alpha)$.

B.3 Zero-Knowledge and Simulation

To prove that Dilithium-G does not leak knowledge of the secret keys $\mathbf{s}_1, \mathbf{s}_2$, we need to show that the distribution of $(\mathbf{z}_1, \mathbf{z}_2, c)$ after Step 10 is independent of $\mathbf{s}_1, \mathbf{s}_2$. For this, we use the following lemma which is implicit in the main result of [37].

Lemma B.1. *Let $T = \max_c \|c(\mathbf{s}_1, \mathbf{s}_1)^T\|$ and M be a positive integer. If $\sigma = \kappa T$ and λ is a positive real number such that $e^{\lambda/\kappa + 1/(2\kappa^2)} \leq M$, the statistical distance between $(\mathbf{z}_1, \mathbf{z}_2, c)$ and $D_\sigma^\ell \times D_\sigma^k \times B_{60}$ after Step 10 is at most $2\exp(-\lambda^2/2)$ and the expected number of iterations necessary to output one sample is M.*

Table 2: Parameters for Dilithium-G.

	weak	medium	recommended	very high
modulus q	8380417	8380417	8380417	8380417
d (dropped bits in the public key)	11	11	11	11
weight of c	60	60	60	60
max secret singular value S	230	225	210	145
$\sigma \approx 11 * S * \sqrt{60}$	19600	19200	17900	12400
$\alpha = (q-1)/512$	16368	16368	16368	16368
(k, ℓ)	(2, 2)	(3, 3)	(4, 4)	(5, 5)
η	7	6	5	3
$B \approx$ as in Eq. (6)	$750K$	$904K$	$990K$	$870K$
M (and # of repetitions)	3	3	3	3
pk size (bytes)	800	1184	1568	1952
sig size (bytes)	1250	1850	2435	2950
BKZ block-size b to break SIS	210	375	555	780
Best Known Classical bit-cost	61	109	162	228
Best Known Quantum bit-cost	55	99	147	206
Best Plausible bit-cost	43	77	115	161
BKZ block-size b to break LWE	205	345	485	595
Best Known Classical bit-cost	59	100	142	174
Best Known Quantum bit-cost	54	91	129	158
Best Plausible bit-cost	42	71	101	124

To see how this lemma is used for setting our parameters, we will use as an example the "recommended" set of parameters from Table 2. The secret key $(\mathbf{s}_1, \mathbf{s}_2)$ is generated at random with each coefficient uniformly distributed between -5 and 5 until the maximum singular value of $\begin{bmatrix} \mathrm{rot}(\mathbf{s}_1) \\ \mathrm{rot}(\mathbf{s}_2) \end{bmatrix} \in \mathbb{Z}^{256(k+\ell)\times 256}$ is less than 210. This implies that the T in Lemma B.1 is $210 \cdot \sqrt{60} \approx 1626$. If we, for example, set $\sigma = 11T$ and $\lambda = 12$, then we obtain that the statistical distance is approximately e^{-72} and $e^{\lambda/\kappa+1(2\kappa^2)} < 3$. So we can set $M = 3$.

To simulate a signature, we sample $(\mathbf{z}_1, \mathbf{z}_2, c) \leftarrow D_\sigma^\ell \times D_\sigma^k \times B_{60}$, then compute $\mathbf{w} := \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}$. Once we have $\mathbf{w}$, we can compute the hint $\mathbf{h}$. We then program $c := \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$. Unless the value for $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ has already been set, the programming step is valid and our simulation is complete. If $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$ has been previously assigned a value, then it must be that

$$\mathsf{HighBits}_q(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}, \alpha) = \mathbf{w}_1.$$

Since the codomain of the above equation, as a function of $\mathbf{z}_1, \mathbf{z}_2$, is extremely large (greater than 2^{4000} for all the parameters in Table 2), if $\mathsf{HighBits}_q(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}, \alpha)$ hits an already existing value, it implies that $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2$ is not uniformly-distributed in R_q^k when $(\mathbf{z}_1, \mathbf{z}_2) \leftarrow D_\sigma^\ell \times D_\sigma^k$. Using the search-to-decision reduction for Module-LWE [35] (which is applicable because our σ is sufficiently high) it implies that the search Module-LWE problem is hard. [8] Thus we can assume that the programming of H is always valid.

[8] The σ is so high that, with a little technical analysis, it should be possible to prove that $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2$ is indeed statistically-close to uniform.

B.4 Security Proof

Throughout the proof we will make the assumption (which is favorable to the adversary) that he gets $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ as the public key. In the real scheme, he only gets the higher order bits of $\mathbf{t}$. In practice, therefore, the scheme may be even more difficult for the adversary to break.

Lemma B.2. *Forging a signature implies finding* $\mathbf{u}_1, \mathbf{u}_2, u_3$ *such that* $\|\mathbf{u}_1, \mathbf{u}_2\| \le 2B$ *and* $\|u_3\|_\infty \le 2$ *such that* $\mathbf{A}\mathbf{u}_1 + \mathbf{u}_2 = u_3\mathbf{t}_1 \cdot 2^d$ *and* $(\mathbf{u}_1, \mathbf{u}_2, u_3) \neq 0$.

Proof. The public key is set to be $(\mathbf{A}, \mathbf{t})$. Signature queries by the adversary are created via the procedure described in Section 4.3. We now describe our extractor when the Adversary produces a winning query. If the Adversary creates a signature $(\mathbf{z}_1, \mathbf{h}, c)$ for a message μ, then he must have queried

$$\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu) = c, \tag{8}$$

where $\mathbf{w}_1 = \mathsf{UseGHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z}_1 - \mathbf{t}_1 c \cdot 2^d, \alpha)$, either directly by querying H or indirectly during a signing query.

Case 1: If c was queried during a signing query, then the reduction already knows another $(\mathbf{z}_1', \mathbf{h}', c)$ (and the associated $\mathbf{w}_1'$) for a message μ' such that $\mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1', \mu') = c = \mathrm{H}(\rho, \mathbf{t}_1, \mathbf{w}_1, \mu)$. This implies that $\mu = \mu'$ and $\mathbf{w}_1 = \mathbf{w}_1'$, or else we found a second pre-image for c. Since $\mathbf{w}_1 = \mathbf{w}_1'$, it must be that

$$\begin{cases} \mathsf{UseGHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z}_1 - \mathbf{t}_1 c \cdot 2^d, \alpha) = \mathbf{w}_1 \\ \mathsf{UseGHint}_q(\mathbf{h}', \mathbf{A}\mathbf{z}_1' - \mathbf{t}_1 c \cdot 2^d, \alpha) = \mathbf{w}_1 \end{cases}.$$

If $\mathbf{z}_1 = \mathbf{z}_1'$, then Lemma A.3 states that we must have $\mathbf{h} = \mathbf{h}'$, and so the adversary has not found a new signature. Thus it must be that

$\mathbf{z}_1 \neq \mathbf{z}_1'$. If we define

$$\begin{cases} \mathbf{u} = \mathbf{A}\mathbf{z}_1 - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot \alpha \\ \mathbf{u}' = \mathbf{A}\mathbf{z}_1' - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot \alpha \end{cases},$$

then subtracting we obtain

$$\mathbf{A}(\mathbf{z}_1 - \mathbf{z}_1') - (\mathbf{u} - \mathbf{u}') = \mathbf{0}.$$

From the verification equation, we know that $\|(\mathbf{z}_1, \mathbf{u})\| \leq B$ and $\|(\mathbf{z}_1', \mathbf{u}')\| \leq B$, and therefore $\|(\mathbf{z}_1 - \mathbf{z}_1', \mathbf{u} - \mathbf{u}')\| \leq 2B$.

Case 2: We now handle the case where the query in Eq. (8) was done directly to H. A standard forking lemma argument shows that the reduction can then extract two signatures $(\mathbf{z}_1, \mathbf{h}, c)$ and $(\mathbf{z}_1', \mathbf{h}', c')$ for $c \neq c'$ such that

$$\begin{cases} \mathsf{UseGHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z}_1 - \mathbf{t}_1 c \cdot 2^d, \alpha) = \mathbf{w}_1 \\ \mathsf{UseGHint}_q(\mathbf{h}', \mathbf{A}\mathbf{z}_1' - \mathbf{t}_1 c' \cdot 2^d, \alpha) = \mathbf{w}_1 \end{cases}.$$

If we define

$$\begin{cases} \mathbf{u} = \mathbf{A}\mathbf{z}_1 - \mathbf{t}_1 c \cdot 2^d - \mathbf{w}_1 \cdot \alpha \\ \mathbf{u}' = \mathbf{A}\mathbf{z}_1' - \mathbf{t}_1 c' \cdot 2^d - \mathbf{w}_1 \cdot \alpha \end{cases},$$

then subtracting we obtain

$$\mathbf{A}(\mathbf{z}_1 - \mathbf{z}_1') - (\mathbf{u} - \mathbf{u}') = \mathbf{t}_1 \cdot 2^d \cdot (c - c').$$

From the verification equation, we know that $\|(\mathbf{z}_1, \mathbf{u})\| \leq B$ and $\|(\mathbf{z}_1', \mathbf{u}')\| \leq B$, and therefore $\|(\mathbf{z}_1 - \mathbf{z}_1', \mathbf{u} - \mathbf{u}')\| \leq 2B$. □

B.5 Representing Gaussians

The output of the signature scheme (Algorithm 14) is a polynomial vector $\mathbf{z}_1$ each of whose coefficients is distributed as a discrete Gaussian over $\mathbb{Z}$ with standard deviation σ, and a "hint" vector $\mathbf{h}$ consisting of "carry" bits when adding a discrete Gaussian to a uniformly-distributed element in $\mathbb{Z}_q$. In both cases, the integers are distributed according to probability distributions that are more weighted towards elements with small norms, and therefore it is not optimal to simply represent integers with their binary representations. The optimal representation would be via Huffman encoding (as in [25]), and what we propose is essentially a simplified version that is nearly optimal.

In the case of coefficients of $\mathbf{z}_1$, we write each coefficient

$$z \bmod^{\pm} q = z_1 \cdot 2^\delta + z_0$$

where $z_0 = z \bmod^{\pm} q$. Since the expected absolute value of z is approximately σ, when 2^δ is also approximately σ, then the value of z_0 is close to being uniform between $-2^{\delta-1}$ and $2^{\delta-1}$. The distribution of z_1, however, has tails that decrease very fast. A simple and close to optimal representation of z is therefore to send z_0 uncompressed (which requires δ bits) and then encode z_1 using the prefix-free encoding of Table 3.

When $\sigma = 2^\delta$, the above compression requires on average approximately 2.25 bits to represent each coefficient of z_1. Thus the total representation of z is on average $\delta + 2.25$ bits.

The coefficients of $\mathbf{h}$ have a very similar distribution to the coefficients z_1 when $\alpha \approx \sigma$. Therefore we can use the same encoding for this vector as well. To get a rough approximation of the number of bits required to represent the signature $(\mathbf{z}_1, \mathbf{h}, c)$ is therefore $(2.25 + \delta) \cdot 256 \cdot \ell + 2.5 \cdot 256 \cdot k + 256$. The exact number will differ because we do not necessarily take the value of σ which is equal to α and 2^δ. The signature sizes in Table 2 represent upper bounds that are exceeded less than 1% of the time in practice. If one would like to have a strict limit on the signature size, then the signer could check to make sure that the signature has length less than this bound before outputting the signature. If the length is greater, then the signer can simply restart the procedure anew.

Table 3: Prefix-free Encoding for the high-order integers of the coefficients of $\mathbf{z}_1$ and for the coefficients of $\mathbf{h}$.

Integer	Representation	Bits
0	00	2
1	01	2
−1	10	2
k (≥ 2)	$110^{2k-4}1$	$2k-1$
$-k$ (≤ -2)	$110^{2k-3}1$	$2k$